VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

VYTVOŘENÍ INTERAKTIVNÍCH POMŮCEK Z OBLASTI 2D POČÍTAČOVÉ GRAFIKY

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

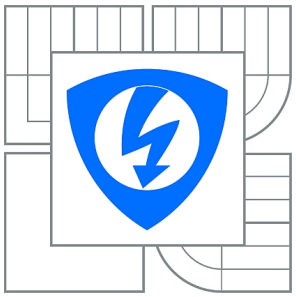
AUTOR PRÁCE
AUTHOR

Bc. JAKUB MALINA

BRNO 2012

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV TELEKOMUNIKACÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF TELECOMMUNICATIONS

VYTVOŘENÍ INTERAKTIVNÍCH POMŮCEK Z OBLASTI 2D POČÍTAČOVÉ GRAFIKY

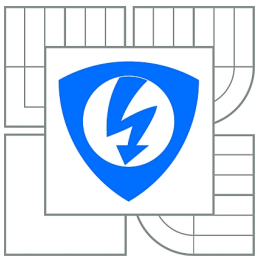
TEACHING AIDS FOR 2D COMPUTER GRAPHICS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. JAKUB MALINA
AUTHOR

VEDOUCÍ PRÁCE — Mgr. PAVEL RAJMIC, Ph.D.
SUPERVISOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky a komunikačních technologií

Ústav telekomunikací

Diplomová práce

magisterský navazující studijní obor
Telekomunikační a informační technika

Student:	Bc. Jakub Malina	***ID:***	106613
Ročník:	2	***Akademický rok:***	2011/2012

NÁZEV TÉMATU:

Vytvoření interaktivních pomůcek z oblasti 2D počítačové grafiky

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte základy popisu křivek a jejich zpracování ve 2D. Implementujte a otestujte grafické uživatelské rozhraní pro applety: a/ racionální Bézierova křivka, b/ NURBS křivka a de Boorův algoritmus, c/ adaptivní a neadaptivní rasterizace křivek, d/ rasterizace úsečky, případně kružnice.

DOPORUČENÁ LITERATURA:

[1] Jiří Žára, Bedřich Beneš, Jiří Sochor, Petr Felkel, Moderní počítačová grafika (2. vydání), Computer Press, 2005, ISBN 80-251-0454-0

[2] Les A. Piegl, Wayne Tiller, The NURBS Book. Springer, druhé vydání, 1996, ISBN-10: 3540615458

Termín zadání:	6.2.2012	***Termín odevzdání:***	24.5.2012

Vedoucí práce: Mgr. Pavel Rajmic, Ph.D.
Konzultanti diplomové práce:

prof. Ing. Kamil Vrba, CSc.
Předseda oborové rady

UPOZORNĚNÍ:

ABSTRAKT

V této diplomové práce se budeme zabývat popisem základních vlastností počítačových křivek a jejich praktickou použitelností. Vysvětlíme si, jak lze křivky chápat obecně, co to jsou polynomiální křivky a způsoby napojování. Poté se zaměříme na popis Bézierových křivek, hlavně pak na Bézierovy kubiky. Podrobněji probereme některé stěžejní algoritmy, které se používají pro vykreslování těchto křivek na počítačích, a ukážeme si jejich praktickou implementaci. Poté probereme neuniformní racionální B-spline křivky a De Boorův algoritmus. Nakonec projdeme tématem rasterizace úsečky, silné čáry, kružnice a elipsy. Cílem diplomové práce je vytvoření několika interaktivních appletů, simulujících algoritmy pro rasterizaci a vykreslení křivek probírané v teoretické části. Tyto applety napomůžou snazšímu pochopení teoretických poznatků a zefektivní výuku.

KLÍČOVÁ SLOVA

Počítačové křivky, Bézierovy křivky, de Casteljau, 2D grafika, napojování křivek, polynomiální křivky, stupěň křivky, interpolace, aproximace, applet, NURBS, De Boor, Bresenham, Midpoint, DDA, rasterizace, B-spline, Flex, HTML5, Silverlight, RIA

ABSTRACT

In this master's thesis we focus on the basic properties of computer curves and their practical applicability. We explain how the curve can be understood in general, what are polynomial curves and their composing possibilities. Then we focus on the description of Bézier curves, especially the Bézier cubic. We discuss in more detail some of fundamental algorithms that are used for modelling these curves on computers and then we will show their practical interpretation. Then we explain non uniform rational B-spline curves and De Boor algorithm. In the end we discuss topic rasterization of segment, thick line, circle and ellipse. The aim of master's thesis is the creation of the set of interactive applets, simulating some of the methods and algorithm we discussed in theoretical part. This applets will help facilitate understanding and will make the teaching more effective.

KEYWORDS

Computer curves, Bézier cubic, de Casteljau, 2D graphics, composition of the curve, polynomial curve, degree of the curve, interpolation, approximation, applet, NURBS, De Boor, Bresenham, Midpoint, DDA, rasterization, B-spline, Flex, HTML5, Silverlight, RIA

MALINA, Jakub *Vytvoření interaktivních pomůcek z oblasti 2D počítačové grafiky*: diplomová práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2012. 88 s. Vedoucí práce byl Mgr. Pavel Rajmic, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Vytvoření interaktivních pomůcek z oblasti 2D počítačové grafiky“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

(podpis autora)

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce panu Mgr. Pavlu Rajmicovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

(podpis autora)

OBSAH

SEZNAM OBRÁZKŮ

ÚVOD

Tato práce se věnuje oblasti 2D počítačové grafiky, zejména oblastem jako jsou Bézierovy křivky a způsoby jejich výpočtu, NURBS křivkám a algoritmy pro rasterizaci. Hlavní náplní práce je vytvoření appletů jako pomůcek při výuce počítačové grafiky. Applety napomáhají studentům lépe chápat a zvládat výuku, která může být mnohdy abstraktní a nesnadno zvládnutelná. Jelikož je dnes mnoho učeben vybaveno počítači a projektory, interaktivní applety mohou výuku nejen usnadnit, ale i zpříjemnit. Jejich hlavní přínos spočívá hlavně v grafické názornosti probíraného učiva a k aplikaci jinak nesnadno představitelných matematických rovnic. Přednášející i studenti si mohou applety vyzkoušet, ať už doma či ve škole. Applety mají grafické uživatelské rozhranní, které umožňuje měnit vstupní parametry, na základě kterých můžeme sledovat například rasterizaci úsečky nebo kružnice, vykreslení počítačových křivek např. NURBS, změnu bázových funkcí na základě změny váhového vektoru apod. U každého appletu důkladně rozebereme jak teoretickou část, tak i praktický popis jejich funkce a ovládání.

Tato práce navazuje na mou bakalářskou práci, jejímž výsledkem je soubor interaktivních appletů pro výuku 2D počítačové grafiky. V bakalářské práci jsem se zabýval zejména Bézierovými křivkami, algoritmem de Casteljau a algoritmem pro navýšení stupně křivky. Podobným směrem pokračuji v této navazující diplomové práci, vytvořením dalších interaktivních appletů, které jsou stejně jako u bakalářské práce umístěny na internetu.

Výsledkem této diplomové práce jsou applety pro ukázku rasterizace pomocí Bresenhamova a Midpoint algoritmu, applet pro modelování NURBS, algoritmu de Boor, dále racionálních Bézierových křivek a také applet B-spline bázových funkcí. Applety jsem vytvořil ve vývojovém prostředí Flex, narozdíl od appletů v předchozí bakalářské práci, které jsou naprogramovány v jazyku Java. V kapitole Praktická realizace vysvětlím, proč jsem se takto rozhodl.

V rámci práce se postupně seznámíme s teorií a ukážeme si principy některých algoritmů. V kapitole NURBS se zabýváme neuniformními B-spline křivkami, které nacházejí široké uplatnění v každodenní praxi návrhářů, designerů, konstruktérů a architektů. Rovněž rasterizace je využívaná v každodenní praxi. DDA a Bresenhamův algoritmus jsou dodnes velmi často používanými algoritmy pro rasterizaci.

Struktura práce

První kapitola s názvem *Křivky obecně* vysvětluje, co křivka je, jak ji chápeme v počítačové grafice a její základní vlastnosti. Dalším důležitým tématem v této

kapitole je napojování křivek. Bézierovy křivky jsou tématem druhé kapitoly. Je zde celistvě rozebrána teorie, způsoby jejich výpočtu a jejich aplikace v podobě Bézierových kubik. Třetí kapitola náleží NURBS křivkách a de Boorovu algoritmu a otevírá tak velké téma počítačové grafiky. Čtvrtá kapitola detailně probírá rasterizaci úsečky i jiných tvarů. Jsou zde rozebrány dva důležité algoritmy a u každého je krátce shrnut postup, s jakým pracují. Kapitola pátá se zabývá praktickou realizací appletů. Je zde popsáno, co je „bohatá internetová aplikace“, jak je možno ji realizovat a co je k tomu zapotřebí. Rovněž je zde otevřeno téma technologie Adobe Flex, zda-li a proč má budoucnost a podrobný popis běhových prostředí Adobe Flash a Adobe AIR. Jsou zde také popsány programovací jazyky pro tvorbu appletů a aplikací, a to MXML a ActionScript. Předposlední šestá kapitola s názvem *Výsledky studenstké práce* je shrnutím dosažených výsledků. V závěru stručně popíšeme celou dosavadní práci, čeho jsme dosáhli a čeho se dosáhnout nepodařilo. Uvedeme zde krátké shrnutí celé problematiky.

1 KŘIVKY OBECNĚ

Pokud mluvíme o křivce a jejích vlastnostech, budeme ji chápat fyzikálně jako dráhu pohybujícího se bodu, ať už ve dvourozměrném či trojrozměrném prostoru. V moderní počítačové grafice používáme křivky v mnoha oblastech. Jejich využití je možné nalézt např. v technických oborech, jako jsou strojírenství nebo architektura. Setkáváme se s nimi při modelování ve dvou i ve třech rozměrech, při definici a vykreslování počítačových fontů, při určování dráhy pohybujících se objektů v počítačové animaci a při tvorbě animovaných filmů.[1]

1.1 Křivky a jejich vlastnosti

Pro výpočet aktuální pozice pohybujícího se bodu, který modeluje křivku, používáme tři různá matematická vyjádření: explicitní (1.1), implicitní (1.2) a parametrické (1.5).[1]

$$y = f(x) \tag{1.1}$$

$$F(x, y) = 0 \tag{1.2}$$

Aktuální polohu pohybujícího se bodu dostaneme dosazením parametru t do rovnic (1.3) a (1.4).[1]

$$x = x(t) \tag{1.3}$$

$$y = y(t). \tag{1.4}$$

Postupným promítnutím všech souřadnic pro $t \in \langle t_{min}, t_{max} \rangle$, dostaneme křivku $Q(t)$, která je definována rovnicí (1.5). v praxi je nejčastěji využíván právě matematický zápis pomocí parametrického vyjádření. Jeho výhodou je závislost pouze na parametru t. Díky tomu je možné vyjádřit průběhy, kdy křivka prochází vícekrát (v různých časových okamžicích) stejnými body v prostoru. Může se křížit, uzavřít apod.[1]

$$Q(t) = \begin{cases} x = x(t) \\ y = y(t) \end{cases}, \text{ kde } t \text{ je čas a } t \in \langle 0, 1 \rangle. \tag{1.5}$$

Další důležitou vlastností křivek je jejich polohový vektor $\vec{q}(t)$, který je definovaný pomocí vektorové rovnice

$$\vec{q}(t) = (x(t),\, y(t)). \tag{1.6}$$

Pomocí polohového vektoru jsme schopni popsat polohu tělesa a jeho změnou v čase t lze popsat trajektorii pohybu. [4] Jeho absolutní velikost je rovna vzdálenosti bodu $Q(t)$ od počátku soustavy souřadnic $[0, 0]$[1]

$$\vec{q}(t) = Q(t) - [0, 0]. \tag{1.7}$$

Rovnici tečny, tj. přímky, která se v daném bodě dotýká křivky se vypočítá pomocí bodu a první derivace polohového vektoru v bodě na křivce, který se nazývá tečný nebo směrový vektor přímky. [1]

$$\vec{q}(t_0) = [x'(t_0),\, y'(t_0)] = \left(\frac{dx(t_0)}{dt},\, \frac{dy(t_0)}{dt}\right) \tag{1.8}$$

$$p(s) = Q(t_0) + s\,\vec{q}'(t_0) \tag{1.9}$$

Druhá derivace polohového vektoru nám udává zrychlení, což je změna okamžité rychlosti pohybu po křivce [1]

$$\vec{q''}(t_0) = x'(t_0),\, y''(t_0) = \left(\frac{dx^2(t_0)}{dt},\, \frac{dy^2(t_0)}{dt}\right). \tag{1.10}$$

Bod, ve kterém jsou vektory rychlosti a zrychlení, tedy první a druhé derivace rovnoběžné, se nazývá Inflexní bod I

$$\vec{q}'(t) = k\,\vec{q}'(t), \text{ kde } k \neq 0 \tag{1.11}$$

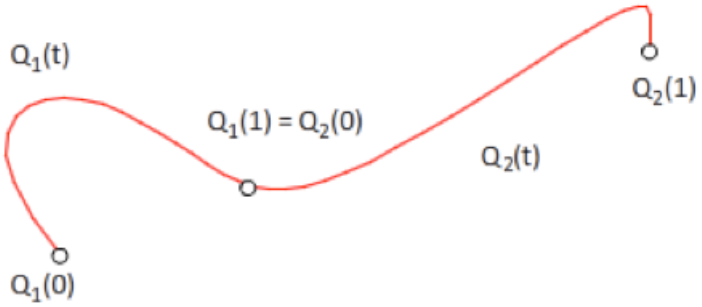
$$I = Q(t), \tag{1.12}$$

kde t je proměnnou času, jejíž hodnota je definována rovnicí (1.11). [1]

1.2 Napojování křivek

V technické praxi se často setkáváme se složitými křivkami, které jsou složeny z několika na sebe napojených křivek. Díky možnosti napojování křivek jsme schopni vykreslovat i velmi složité obrazce. Ukázku napojení několika křivek na sebe můžeme vidět na obrázku 2.1, kde koncový bod jedné křivky je zároveň počátečním bodem křivky následující. Těmto bodům budeme říkat uzly.[1]

Křivky můžeme napojovat různými způsoby. Pokud chceme dosáhnout určitého stupně hladkosti napojení, nebude nás zajímat pouze rovnost počátečního a koncového bodu, ale především tzv. spojitost v uzlu. Říkáme, že dvě křivky napojené na sebe jsou parametricky nebo také C_k spojitě na sebe navázány, pokud v jejich společném uzlu jsou shodné vektory všech derivací až do řádu k, kde pod proměnnou k rozumíme první, druhou až k-tou derivaci polohového vektoru. To znamená,



Obr. 1.1: Napojení dvou křivek.

že k náleží do množiny přirozených čísel. Čím vyšší spojitost je požadována, tím delší dobu, ve smyslu parametru t, se obě křivky k sobě přimykají. Například ze spojitosti třídy C^0 plyne, že bod se pohybuje po křivce, ale v uzlu napojení může náhle změnit směr. Spojitost C^1 nám zase říká, že bod nemůže náhle změnit směr pohybu ani velikost okamžité rychlosti, ale zrychlení změnit může. Při spojitosti C^2 se nezmění směr, okamžitá rychlost ani zrychlení, tj. zakřivení. [1]

Mnohdy se při prvním pohledu na spojení dvou křivek nebo segmentů zdá, že křivky jsou spojeny parametricky, ale při bližší analýze zjistíme, že tečné vektory sice mají stejný směr, ale jejich rychlost a zrychlení se mění. Takové napojení nazýváme geometrická spojitost a značíme ji G^n.[1]

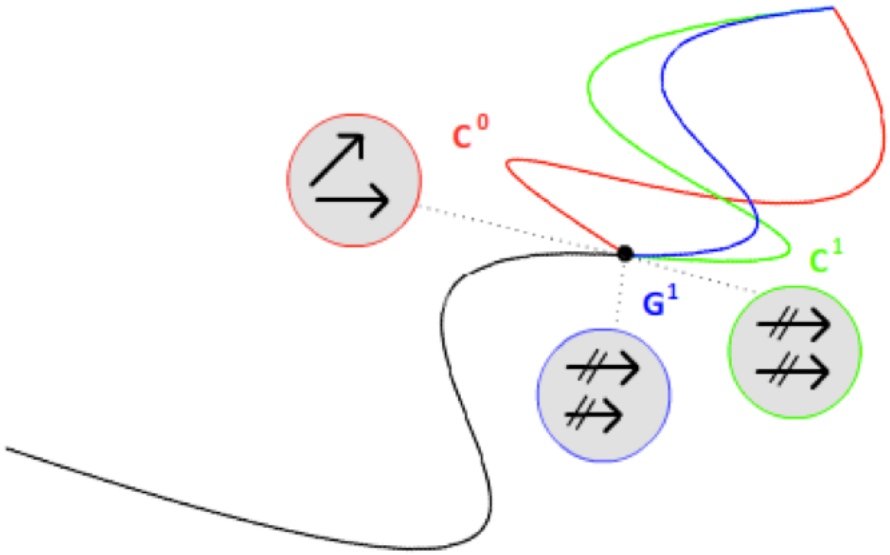
$$q_1(1) = k\, q_1(0), \text{ kde } k > 0 \tag{1.13}$$

Pro geometrickou spojitost třídy G^1, která se používá nejčastěji, je typická různá délka vektorů. Pohybující se bod po křivce tedy nemůže náhle změnit směr, ale rychlost a zrychlení ano. Příklad jednotlivých spojitostí vidíme na obrázku 1.2, kde můžeme vidět grafickou podobnost hladkosti napojení mezi G^1 a C^1. V praxi bývá často snazší realizovat geometrické napojení G^1 než parametrické C^1. Spojitost C^1 implikuje G^1, obráceně však nikoli.[1]

Spojitost G^2 je definována jako shoda prvních křivostí ${}^1k$ v uzlu obou segmentů, kde tzv. první křivost ${}^1k$ se spočítá podle vztahu (1.14).[1]

$$ {}^1k_{Q1}(1) = {}^1k_{Q2}(0) \tag{1.14}$$

$$ {}^1k_{Q1}(t_0) = \sqrt{\frac{\left|\vec{q'}(t_0) \times \vec{q'}(t_0)\right|^2}{\left|\vec{q'}(t_0) \cdot \vec{q''}(t_0)\right|^3}} \tag{1.15}$$



Obr. 1.2: Rozdíly mezi parametrickou spojitostí nultého stupně C^0, prvního stupně C^1 a geometrickou spojitostí prvního stupně G^1.

1.3 Definice polynomiálních křivek

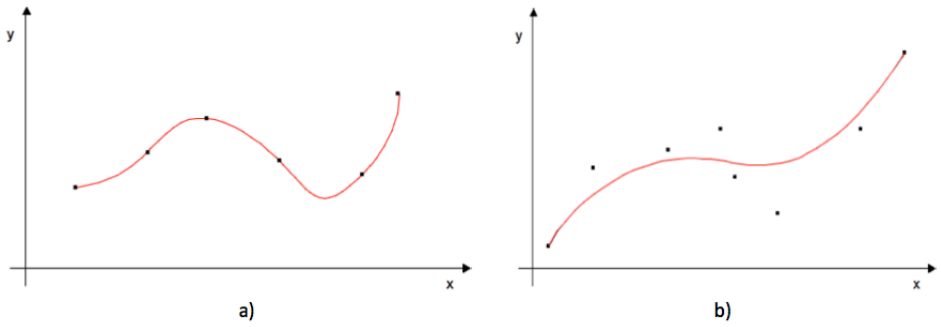
Základním a nejčastěji používaným druhem křivek v počítačové grafice jsou křivky polynomiální, které se dají velice rychle vypočítat a jsou snadno diferencovatelné. Pro snadnou manipulovatelnost, dostatečně širokou škálu tvarů, snadnost výpočtu a možnost zaručit parametrickou spojitost C^2 se používají polynomy 3. stupně, také nazývané kubiky. Křivky vyššího stupně se příliš nepoužívají z důvodu vzniku nežádoucího vlnění a složitosti jejich výpočtu.

Modelování křivek provádíme zvolením několika, čtyř v případě kubik, řídících bodů, na jejichž základě se pomocí matematického výpočtu provede vykreslení křivky. Jinou metodou je určení počátečního a koncového bodu křivky a poté zvolení tečných vektorů v těchto bodech. Při napojování křivek také máme možnost nastavovat hladkost napojení popsaného v kapitole 1.2.

Existují dva základní způsoby interpretace řídících bodů, a to interpolace a aproximace. Při interpolaci generovaná křivka prochází danými body, zatímco při aproximaci je řídícími body tvar křivky určen, ale křivka jimi procházet nemusí.

Obecný tvar parametricky zadané kubiky vypadá takto

$$Q(t) \begin{cases} x = x(t) = a_x t^3 + b_x t^2 + c_x t + d_x \\ y = y(t) = a_y t^3 + b_y t^2 + c_y t + d_y \end{cases}, \tag{1.16}$$



Obr. 1.3: Iterpolační a) a aproximační b) příklady křivek.

to lze maticově zapsat takto

$$Q(t) = \mathbf{TC} = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} a_x & a_y \\ b_x & b_y \\ c_x & c_y \\ d_x & d_y \end{bmatrix} \tag{1.17}$$

Derivací $\mathbf{T}$ získáme směrový vektor

$$\vec{q'}(t) = \frac{d}{dt}Q(t) = \frac{d}{dt}\mathbf{TC} = \begin{bmatrix} 3t^2 & 2t & 1 & 0 \end{bmatrix} \mathbf{C}. \tag{1.18}$$

Matice $\mathbf{C}$ obsahuje parametry, které ovlivňují tvar křivky. Jejich zadávání není příliš intuitivní, a proto neefektivní. Rozepíšeme ji tedy následovně

$$\mathbf{C} = \mathbf{MG}. \tag{1.19}$$

Výslednou křivku získáme vynásobením všech tří matic

$$Q(t) = \mathbf{TMG}. \tag{1.20}$$

Bázová matice $\mathbf{M}$ o velikosti 4×4 je určená danou metodou vykreslení křivek a určuje způsob výpočtu. Pro Bézierovy kubiky vypadá matice takto

$$\mathbf{M} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}. \tag{1.21}$$

Druhá matice, která vznikla rozložením matice $\mathbf{C}$ je matice $\mathbf{G}$ o rozměrech 4×2. Ta se nazývá maticí geometrických podmínek a jsou v ní obsaženy řídící body. Pro vykreslení kubiky tedy stačí zadat souřadnice čtyř bodů $P_1 \dots P_4$ do matice $\mathbf{G}$.

2 BÉZIEROVY KŘIVKY

Mezi nejznámější aproximační křivky v počítačové grafice patří právě Bézierovy křivky. Teoretický základ těchto křivek vytvořil na přelomu 50. a 60. let P. E. Bézier, když vyvíjel programový nástroj UNISURF pro návrh křivek a ploch u francouzské firmy Renault. Jejich častým využitím je definice počítačových fontů. Nejdříve popíšeme obecné Bézierovy křivky n-tého stupně a poté si představíme nejčastěji používanou variantu kubiky.

Pro Bézierovu křivku n-tého stupně definujeme $n+1$ řídících bodů P_k. Bézierovy křivky jsou vystavěny na tzv. Bernsteinových polynomech, které rozebereme v následující kapitole. Obecná definice Bézierovy křivky vypadá takto

$$Q^{\mathrm{BEZ}}(t) = \sum_{k=0}^{n} P_k \cdot B_{k,n}(t), \tag{2.1}$$

kde P_k je $n+1$ řídících bodů a $B_{k,n}(t)$ jsou Bernsteinovy polynomy n-tého stupně. Vypočítáme je následovně

$$B_{k,n}(t) = \begin{pmatrix} n \\ k \end{pmatrix} t^k (1-t)^{n-k}, \tag{2.2}$$

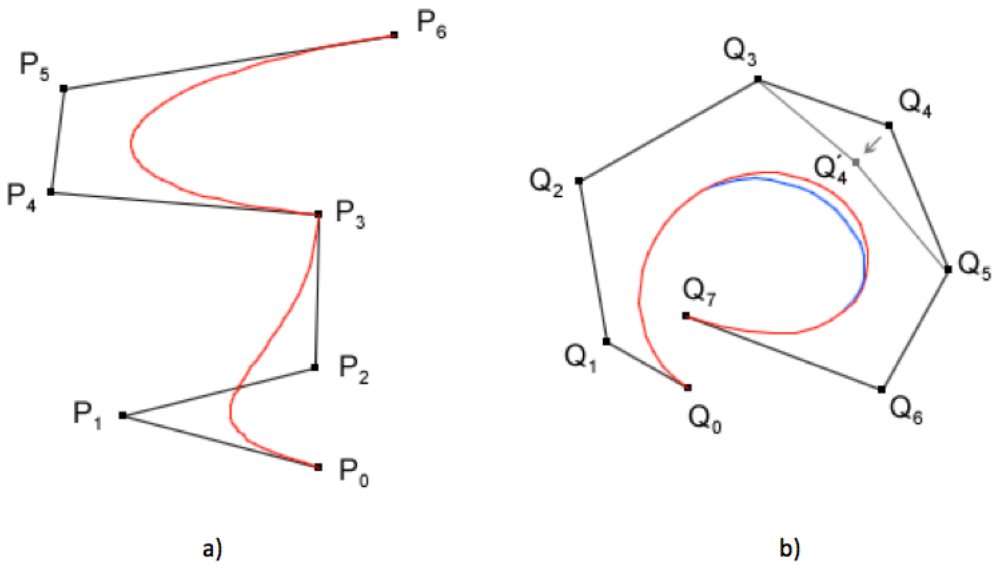
kde $t \in [0,1]$, pro $k = 0, 1, 2, \ldots, n$. Při výpočtu tohoto vztahu je $\begin{pmatrix} n \\ 0 \end{pmatrix} = 1$ a $0^0 = 1$.

$$p(0) = n(P_1 - P_0) \tag{2.3}$$

$$p(1) = n(P_n - P_{n-1}) \tag{2.4}$$

Vložíme-li do vztahu (2.1) parametr $t = 0$, resp. $t = 1$, snadno zjistíme, že křivka prochází prvním a posledním bodem řídícího polygonu, viz výrazy (2.3) a (2.1) pro tečné vektory v krajních bodech. To je jedna ze společných vlastností pro všechny Bézierovy křivky. Dále o nich můžeme říci, že výsledná křivka leží vždy v konvexní obálce bodů řídícího polygonu. [1]

Další vlastností Bézierových křivek je, že při změně polohy jednoho řídícího bodu P_k dojde ke změně tvaru celé křivky a k nutnosti překreslit celý její průběh. Tato vlastnost je jedním z důvodů, proč se vpraxi křivky dělí na segmenty vzájemně na sebe napojených Bézierových křivek nižšího stupně. [1] Praktický příklad můžeme vidět na obrázku 2.1. Vlevo vidíme dvě na sebe napojené křivky, které se vzájemně neovlivňují, zatímco u křivky 7. stupně vpravo je patrné, že změna polohy jednoho z bodů, změní tvar celé křivky.



Obr. 2.1: a) Dvě křivky 3. stupně napojené spojitostí C^0 v bodě P_3, b) křivka 7. stupně před a po změně polohy bodu Q_4.

2.1 Bernsteinovy polynomy

Bézierovy křivky mohou být definovány jako rekurzivní algoritmus, který vynalezl de Casteljau. Je však také nutné mít pro ně explicitní reprezentaci, což nám značně usnadní další teoretický rozvoj.[1] Proto si nyní představíme Bernsteinovy polynomy. Následujících pět rovnic nám definuje základní vlastnosti Bernsteinových polynomů

$$B_{k,n}(0) = \begin{cases} 1 \text{ pro } k = 0 \\ 0 \text{ jinak,} \end{cases} \tag{2.5}$$

$$B_{k,n}(1) = \begin{cases} 1 \text{ pro } k = n \\ 0 \text{ jinak,} \end{cases} \tag{2.6}$$

$$\forall k, n \in N \cup \{0\} \text{ a } t \in \langle 0,\, 1\rangle \text{je} B_{k,n}(t) \geq 0, \tag{2.7}$$

$$\sum_{k=0}^{n}(t)B_{k,n} = 1 \text{ pro } t \in [0, 1], \tag{2.8}$$

$$B_{k,n} = (1-t)B_{k,n}(t) + tB_{k-1,n-1}(t), \tag{2.9}$$

$$B_{k,n-1}(t) = \frac{n-k}{k}B_{k,n}(t) + \frac{k+1}{n}B_{k+1,n}(t), \tag{2.10}$$

$$\frac{d}{dt}B_{k,n}(t) = n\left[B_{k-1,n-1}(t) - B_{k,n-1}(t)\right]. \tag{2.11}$$

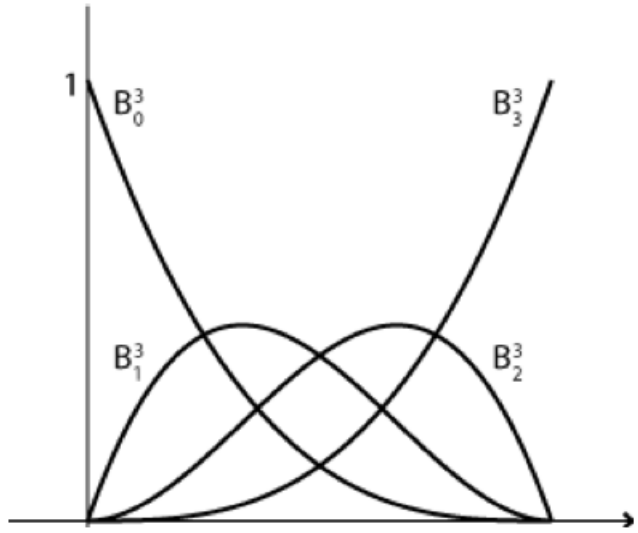
Druhý vztah zaručuje nezápornost Bernsteinových polynomů, tj. jejich funkční hodnoty jsou větší nebo rovny nule. Druhý a třetí pak zaručují, že výsledná křivka

bude vždy ležet v konvexní obálce bodů řídícího polygonu. Čtvrtý vztah je rekurentní definicí Bernsteinova polynomu stupně n pomocí lineární kombinace dvou po sobě následujících Bernsteinových polynomů stupně $n-1$. [1]

Pro praktickou ukázku si definujeme Bernsteinovy polynomy pro kubiku, tedy pro $n = 3$

$$B_{0,3}(t) = (1-t)^3, \tag{2.12}$$
$$B_{1,3}(t) = 3t\,(1-t)^2, \tag{2.13}$$
$$B_{2,3}(t) = 3t^2\,(1-t), \tag{2.14}$$
$$B_{3,3}(t) = t^3. \tag{2.15}$$

Obrázek 2.2 ukazuje skupinu čtyř Bernsteinových polynomů pro třetí stupeň, kubiku. Je dobré poznamenat, že křivky Bernsteinových polynomů jsou v celém intervalu od t_{min} do t_{max} nenegativní, tj. nacházejí se v kladném kvadrantu a nenabývají záporných hodnot.



Obr. 2.2: Bernsteinovy polynomy pro kubiku.[13]

2.2 Bézierovy kubiky

Nejčasteji používaný stupeň polynomu Bézierovy křivky jsou kubiky. Jsou známé využitím při definici počítačových písem a hladkých křivek (smooth curves) v počítačové grafice. Jejich vykreslení pomocí čtyř bodů je intuitivní a rotace či jiné

transformace lze použít na křivku aplikovaním těchto transformací na řídící body konvexní obálky. [1]

Využití takto nízkého stupně křivky má svůj význam v rychlosti výpočtu a v zamezení změny tvaru celé křivky, změnou jednoho bodu. Pokud chceme vykreslit složitější tvary, napojíme několik kubik za sebe jedním z výše popsaných napojení. Takto je ve standartech vektorové grafiky definován pojem „path“ a je prakticky využit v mnoha programech jako Adobe Illustrator, Inkscape nebo právě Adobe Flex. Praktickou ukázkou ve Flex se budeme zabývat v části „Praktická část“.

Bézierovy kubiky jsou využity i v animaci. Program Adobe Flash je používá například pro pohyb objektů. v 3D animacích jsou využity k definici 3D paths.

Písma TrueType používají kvadratické Bézierovy křivky, zatímco PostScript a SVG používají Bézierovy kubiky. OpenType používá jak kubiky tak kvadratické Bézierovy křivky v závislosti na typu písma.

K vykreslení Bézierovy kubiky můžeme využít tří způsobů: matici, algoritmus de Casteljau s konstantním krokem k anebo dělení křivky pomocí algoritmu de Casteljau a následnou aproximací vypočtených bodů na křivce. Všechny způsoby výpočtu rozebereme v následujících kapitolách.

2.3 Výpočet řídících bodů pomocí matice

Rovnice pro výpočet Bézierových kubik je dána součinem tří matic

$$Q(t) = \mathbf{TMG} = \begin{bmatrix} t^3\, t^2\, t\, 1 \end{bmatrix} = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}. \tag{2.16}$$

Tečné vektory v počátečním a koncovém bodě mají tvar

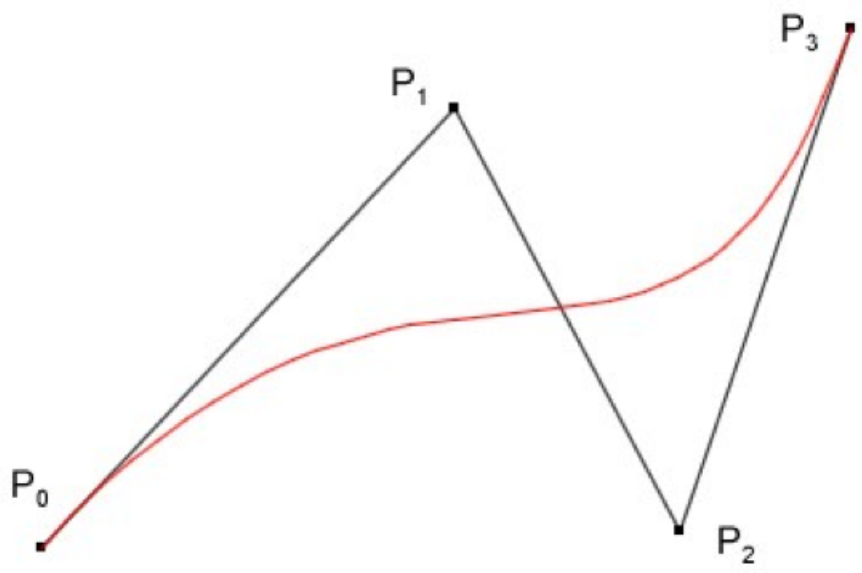
$$p(0) = 3(P_1 - P_0) \tag{2.17}$$

$$p(1) = 3(P_3 - P_2). \tag{2.18}$$

Nyní si ukážeme, jak vypadá Bézierova křivka třetího stupně. Na obrázku 2.3 je červenou čárou vykreslena křivka a černou čárou znázorněna její konvexní obálka.

2.4 Algoritmus de Casteljau

Algoritmus popsaný v této kapitole patří mezi nejzákladnější výpočetní techniky pro křivky. Jeho hlavní přínos spočívá v jednoduchosti a velice intuitivní souhře



Obr. 2.3: Bézierova křivka třetího stupně s řídícími body a konvexní obálkou.

mezi geometrickou konstrukcí a algebraickými výpočty. Díky tomuto algoritmu jsme schopni určit pozici bodu v určitém časovém okamžiku v intervalu od t_{min} do t_{max}. Praktické využití nacházíme například při vykreslování Bézierových křivek. [1]

Algoritmus de Casteljau zapíšeme rekurzivně pomocí vztahu [1]

$$P_i^j(t) = (1-t)P_i^{j-1} + tP_{i+1}^{j-1}, \tag{2.19}$$

kde j je iterační proměnnou cyklu a i je pořadí bodu v daném cyklu.

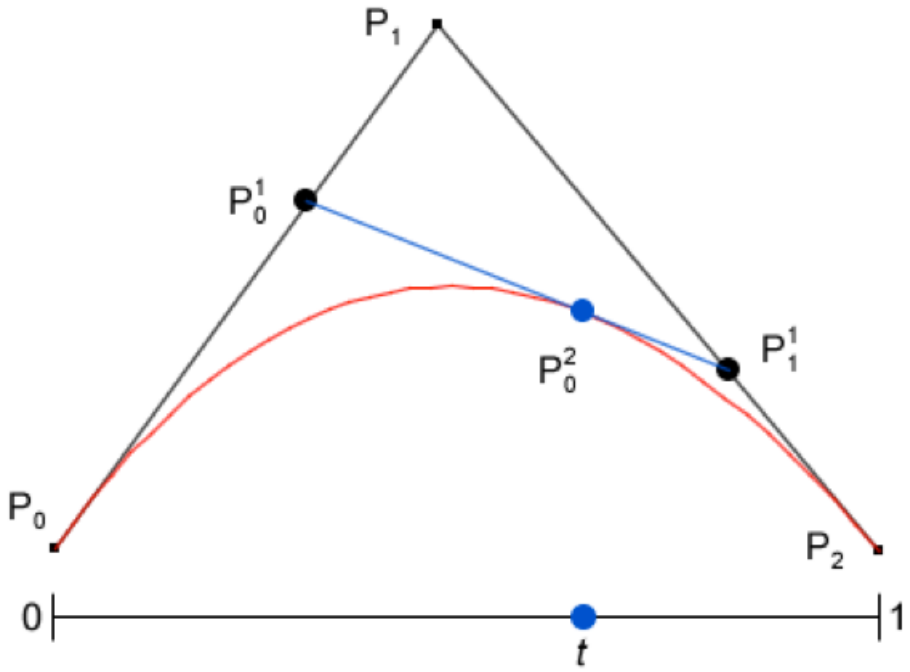
Jako příklad vytvoříme jednoduchou konstrukci pro vykreslení paraboly pomocí kvadratické Bézierovy křivky. Zvolíme si tři řídící body P_0, P_1 a P_2 a parametr t, který náleží do množiny reálných čísel. Následující tři rovnice nám pomůžou k určení pozice bodů P_0^1, P_1^1 a P_0^2

$$P_0^1(t) = (1-t)P_0 + tP_1, \tag{2.20}$$
$$P_1^1(t) = (1-t)P_1 + tP_2, \tag{2.21}$$
$$P_0^2(t) = (1-t)P_0^1(t) + tP_1^1(t). \tag{2.22}$$

Vložením rovnic (2.20) a (2.21) do (2.22) dostaneme rovnici pro okamžitý výpočet bodu P_0^2

$$P_0^2(t) = (1-t)^2P_0 + 2t(1-t)P_1 + t^2P_2. \tag{2.23}$$

Opakovaným výpočtem (2.23) pro různé $t \in \langle 0, 1 \rangle$ dostáváme parabalu vykreslenou pohybujícím se bodem P_0^2 po dráze v zadané konvexní obálce.

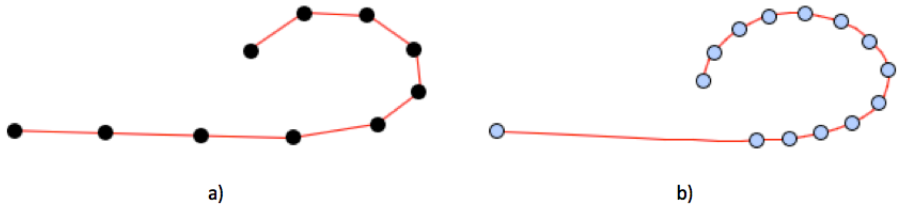


Obr. 2.4: Parabola sestrojená konstrukcí algoritmu de Castejlau.

Na tomto obrázku vidíme princip vykreslení paraboly. Na úsečce mezi dvěma řídícími body vytvoříme nový bod, který spojíme úsečkou s dalším nově vytvořeným bodem mezi dalšími dvěma řídícími body. Tento cyklus probíhá tak dlouho, dokud nevytvoříme jednu jedinou úsečku, která je na obrázku vyznačena modře a na níž leží hledaný bod. V případě tří řídících bodů, tedy pro $n = 2$, vytvoříme poslední úsečku už ve druhém kroku, nicméně můžeme si představit, že stejným principem vykreslujeme křivku s mnohem více řídícími body. [3]

Při vykreslování bodu na křivce můžeme použít dva postupy. Jedním z nich je rozdělení intervalu času od t_{min} do t_{max} s konstantním krokem k, např. $k = 0,05$. Nevýhodou tohoto způsobu je, že nehledě na rovnost či zakřivení křivky je krok k vždy stejný, a proto dochází k nahrazení rovné části mnoha úsečkami, zatímco zakřivená část je aproximována příliš hrubě.

Na obrázku 2.5 je zřejmé, že vykreslení křivky s konstantním krokem k není nejefektivnější způsob. Obrázek vlevo ukazuje vykreslení křivky konstantním krokem k, který je znázorněn černými body. Křivka je aproximována stejně v zakulacené části jako v části rovné. Pravý obrázek ilustruje efektivní způsob vykreslení, kde



Obr. 2.5: Ukázka vykreslení křivky a) konstantním krokem k, b) metodou postupného dělení za pomoci algoritmu de Casteljau.

zakulacená část je aproximována více úsečkami než její rovná část. Výhodou je, že tato metoda generuje menší množství dat než metoda s konstantním krokem k.

2.5 Vykreslování křivky postupným dělením

Jiným způsobem vykreslování je postupné dělení křivky na dvě poloviny. Pokud je jedna z polovin dostatečně rovná, tj. vyhovuje předdefinovaným mezím, pak se aproximuje jednou úsečkou. Pokud není, pokračuje se na ní v dělení. Kritériem pro dostatečně rovnou část může být tloušťka čáry nebo délka úhlopříčky pixelu. v takovém případě je již další dělení zbytečné.

Postup pro rozdělení křivky na dvě poloviny bude pracovat s body $P_1, \ldots, P_3$, $L_1, \ldots, L_3$ a $R_1, \ldots, R_3$, kdy body P_n jsou řídící body původní křivky, body L_n jsou nové body levé poloviny původní křivky a body P_n jsou nové body pravé poloviny původní křivky.

1. Vytvoříme bod L_1 a přiřadíme mu souřadnice P_1,
2. $L_2 = \frac{(P_1+P_2)}{2}$,
3. do pomocné proměnné tmp uložíme $tmp = \frac{(P_2+P_3)}{2}$,
4. $L_3 = \frac{(L_2+tmp)}{2}$,
5. $R_4 = P_4$
6. $R_3 = \frac{(P_3+P_4)}{2}$,
7. $R_2 = \frac{(tmp+R_3)}{2}$,
8. $L_4 = R_1 = \frac{(L_3+R_2)}{2}$.

2.6 Racionální Bézierovy křivky

Bézierovy křivky mají velkou nevýhodu a to tu, že pomocí nich nejsme schopni sestrojit ani jednoduché tvary a kuželosečky, jako jsou kružnice, elipsa apod. Snaha

vyřešit tento nedostatek vede k použití konceptu váhy u každého řídícího bodu. Řídící bod, který má definovanou váhu w tak k sobě přitahuje křivku určitou silou. Na problematiku by se dalo dívat i tak, že bod s vyšší váhou má vyšší důležitost a křivku v daném místě ovlivňuje více než body s nižší váhou. Racionální Bézierova křivka je pak dána vztahem

$$Q^{BEZ}_{rational} = \sum_{k=0}^{n} \frac{w_k P_k B_{k,n}(t)}{\sum_{k=0}^{n} w_k B_{k,n}(t)}. \tag{2.24}$$

Nyní již můžeme modelovat kuželosečky přesně. Je zde však ještě jedna nevýhoda. Racionální křivky obecně trpí tím, že nejsou invariantní vůči perspektivnímu promítání. To by se projevilo například v 3D CAD/CAM systémech. Tuto nepříjemnost však řeší ještě obecnější křivku NURBS.

3 NURBS KŘIVKY

Neuniformní racionální B-spline křivky (zkráceně NURBS) jsou zobecněním tzv. B-spline křivek. Termín neuniformní znamená, že vzdálenosti uzlů ve smyslu parametru t nemusí být u těchto křivek konstantní tak, jako například u Bézierových křivek. NURBS křivka je určena $n+1$ body P_i, kde $i = 0, \ldots, n$ řídícího polygonu, řádem B-spline k a uzlovým vektorem U délky $n+k+1$. Uzlový vektor je pak tvořen posloupností neklesajících reálných čísel, což jsou uzlové hodnoty $t_0 \leq t_1 \leq y \ldots, \leq tn+k$. Hodnoty se mohou i opakovat. Křivka NURBS je určena vztahem [1]

$$Q(t) = \frac{\sum_{i=0}^{n} w_i P_i N_{i,k}(t)}{\sum_{i=0}^{n} w_i N_{i,k}(t)}, \tag{3.1}$$

kde w_i je váha i-tého bodu řídícího polygonu a $N_{i,k}$ jsou normalizované B-spline bázové funkce definované obecně takto [1]

$$N_{i,k}(t) = \frac{t - t_i}{t_{i+k-1} - t_i} N_{i,k-1}(t) + \frac{t_{i+k} - t}{t_{i+k} - t_{i+1}} N_{i+1,k-1}(t), \tag{3.2}$$

pro $t_i < t_{i+1+k}$, kde $0 \leq i \ \leq n$. Pro první stupeň B-spline pak platí [1]

$$N_{i,1}(t) = \begin{cases} 1 & \text{pro } t_i \leq t < t_{i+1} \\ 0 & \text{jinak} \end{cases} . \tag{3.3}$$

Jiný zápis pro křivku využívá racionální B-spline báze [1]

$$R_{i,k} = \frac{w_i N_{i,k}(t)}{\sum_{j=0}^{n} w_j N_{j,k}(t)}. \tag{3.4}$$

Nyní můžeme rovnici (3.1) zapsat zjednodušeně [1]

$$Q(t) = \sum_{i=0}^{n} P_i R_{i,k}(t). \tag{3.5}$$

U Bézierových křivek jsme si definovali jejich explicitní vyjádření v podobě Bernsteinových polynomů, NURBS křivky mají následující vlastnosti:

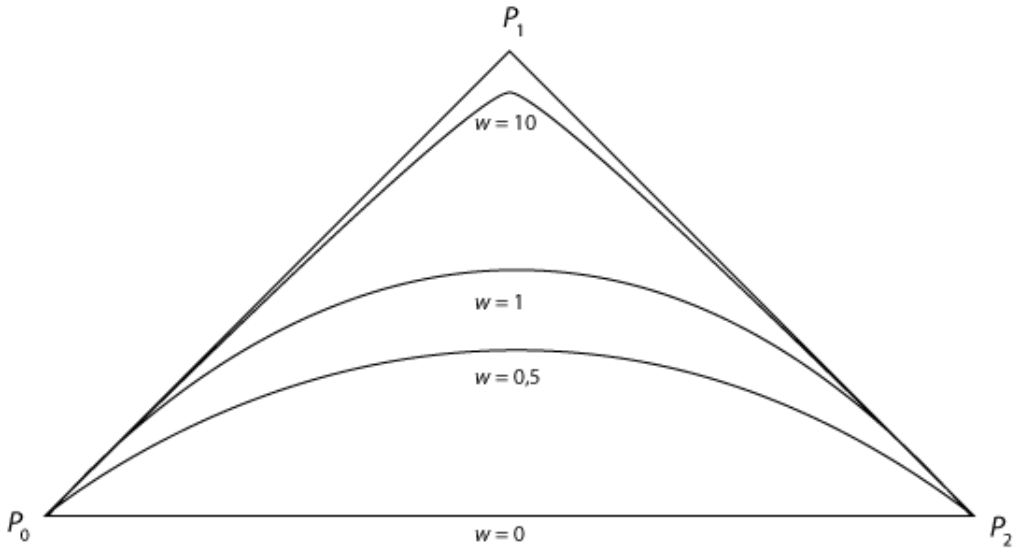
1. $\forall\ i, n \in N\,\{0\}$ a $t \in \langle 0,\, 1\rangle$ je $N_i^n(t) \geq 0$.
2. $\sum_{i=0}^{n} R_i^n(t) = 1$ pro $t \in \langle 0,\, 1\rangle$.
3. $N_{i,k}(t) = \frac{t-t_i}{t_{i+k-1}-t_i} N_{i,k-1}(t) + \frac{t_{i+k}-t}{t_{i+k}-t_{i+1}} N_{i+1,k-1}(t)$ pro $t_i < t_{i+1+k}$,$0 \leq i \ \leq n$.

První vztah zaručuje nezápornost polynomů B-spline báze. První a druhý pak zaručují, že výsledná křivka bude ležet v konvexní obálce řídícího polygonu. Třetí vztah je rekurentní definicí bázového polynomu řádu k a zároveň je základem de Boorova algoritmu pro výpočet křivky NURBS. [1]

Dále jsou křivky NURBS invariantní vůči transformacím a vůči rovnoběžnému a středovému promítání. Pomocí váhových koeficientů w_i můžeme přesně vyjádřit kuželosečky jako podíl polynomů. Pro uzlový vektor o celkové délce $2k$ v následujícím tvaru, kdy polovina vektoru U o délce k je nulová a druhá polovina má všechny uzlové body rovny jedné, a pro hodnoty váhových koeficientů $w_i = 1$ pro všechny body křivky je NURBS Bézierovou křivkou a B-Spline bázové funkce rovny Bernsteinovým polynomům. [1]

$$U = \left\{ \underbrace{0, 0, \ldots, 0}_{k}, \underbrace{1, 1, \ldots, 1}_{k} \right\} \tag{3.6}$$

Pojďme si ukázat vliv váhové koeficientu w_1 bodu P_1 na tvar vykreslení racionální B-spline křivky. Křivka začíná v bodě P_0 a končí v bodě P_2. Její tvar určuje konvexní obálka definovaná zejména bodem bodem P_1. Můžeme říci, že křivka se bude nacházet uvnitř konvexní obálky a její tvar bude ovlivňován váhovým koeficientem bodu P_1 tak, že pro $w_1 = 0$ bude křivka úsečkou a pro $w_1 = \infty$ bude její tvar kopírovat konvexní obálku. [1]

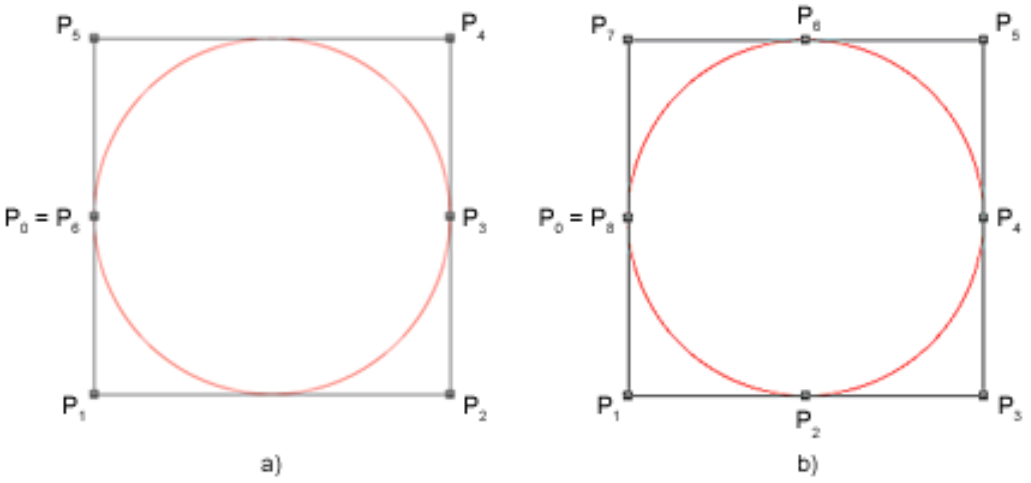


Obr. 3.1: Vlih váhy bodu P_1 na racionální B-spline.

3.1 Praktické příklady NURBS křivek

Nejlépe můžeme pochopit problematiku neuniformních racionálních B-Spline křivek pochopit na praktických příkladech. Bylo řečeno, že pomocí NURBS křivek lze vykreslit kuželosečky, pojďme si ukázat, jak lze vykreslit kružnici. Způsobů vykreslení

kružnice je jistě více, ale uveďme si dva způsoby, které jsou nejvýhodnější pro návrh a použití.



Obr. 3.2: Příklady kružnic pomocí NURBS a) sedmibodová b) devítibodová.

První příklad vlevo zobrazuje NURBS křivku sestrojenou pomocí sedmi bodů, přičemž body P_0 a P_6 mají stejné souřadnice. Je zřejmé, že pohybem bodů P_1 a P_2 neovlivníme horní polovinu křivky, ale pouze spodní, a to stejné platí naopak pro body v horních rozích, tedy P_1 a P_2. Váhový vektor W_{K1} a uzlový vektor U_{K1} jsou definovány pro tuto křivku následovně:

$$W_{K1} = \left\{1, \frac{1}{2}, \frac{1}{2}, 1, \frac{1}{2}, \frac{1}{2}, 1\right\} \tag{3.7}$$

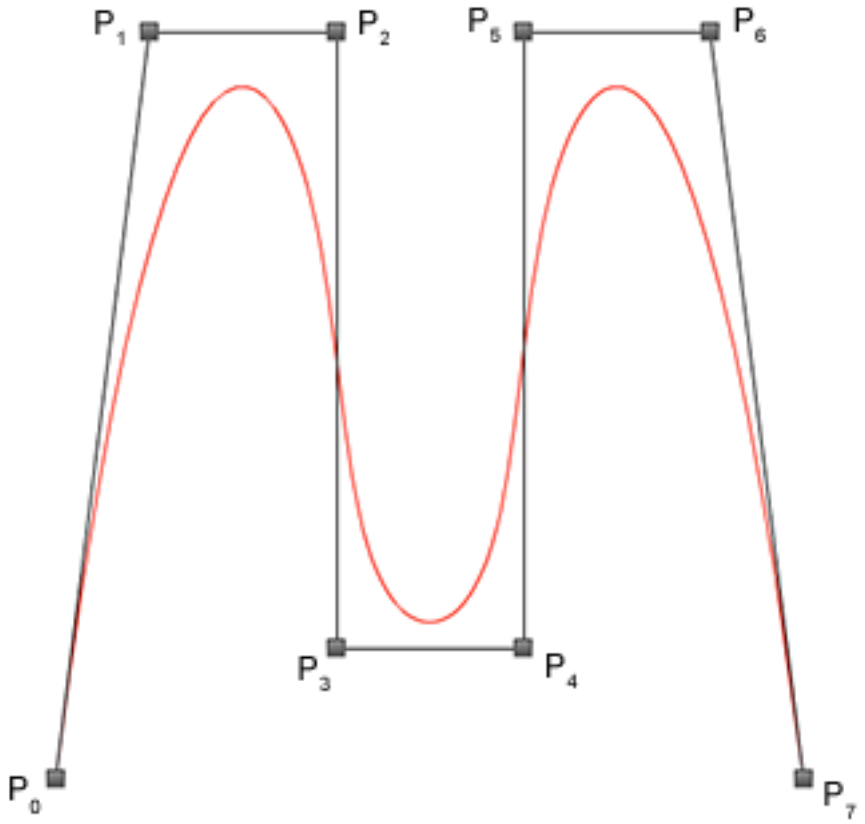
$$U_{K1} = \left\{0, 0, 0, \frac{1}{4}, \frac{1}{2}, \frac{1}{2}, \frac{3}{4}, 1, 1, 1\right\} \tag{3.8}$$

Křivka vpravo je reprezentována devíti body s váhami W_{K2} a uzly U_{K2}

$$W_{K2} = \left\{1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1, \frac{\sqrt{2}}{2}, 1\right\} \tag{3.9}$$

$$U_{K2} = \left\{0, 0, 0, \frac{1}{4}, \frac{1}{4}, \frac{2}{4}, \frac{2}{4}, \frac{3}{4}, \frac{3}{4}, 1, 1, 1\right\} \tag{3.10}$$

Obě kružnice jsou tvořeny křivkami druhého stupně, tedy pro $k = 3$. To se v tomto případě dá zjistit z délky uzlového vektoru a počtu nulových a jedničkových uzlových bodů na začátku a na konci uzlového vektoru. Kružnice s devíti body se nám za cenu většího počtu řídících bodů odmění větší varibialitou, neboť je tvořena čtyřmi čtvrtkruhy při jejichž editaci neovlivníme zbylé tři.



Obr. 3.3: Příklad volného tvaru NURBS.

Složitějším tvarem je křivka vlny na obrázku 3.3 tvořena pěti křivkami třetího stupně. Pro její vykreslení jsou dány tyto parametry:

$$W_{K2} = \{1, 1, 1, 1, 1, 1, 1, 1\} \tag{3.11}$$

$$U_{K2} = \left\{0, 0, 0, 0, \frac{1}{5}, \frac{2}{5}, \frac{3}{5}, \frac{4}{5}, 1, 1, 1, 1\right\} \tag{3.12}$$

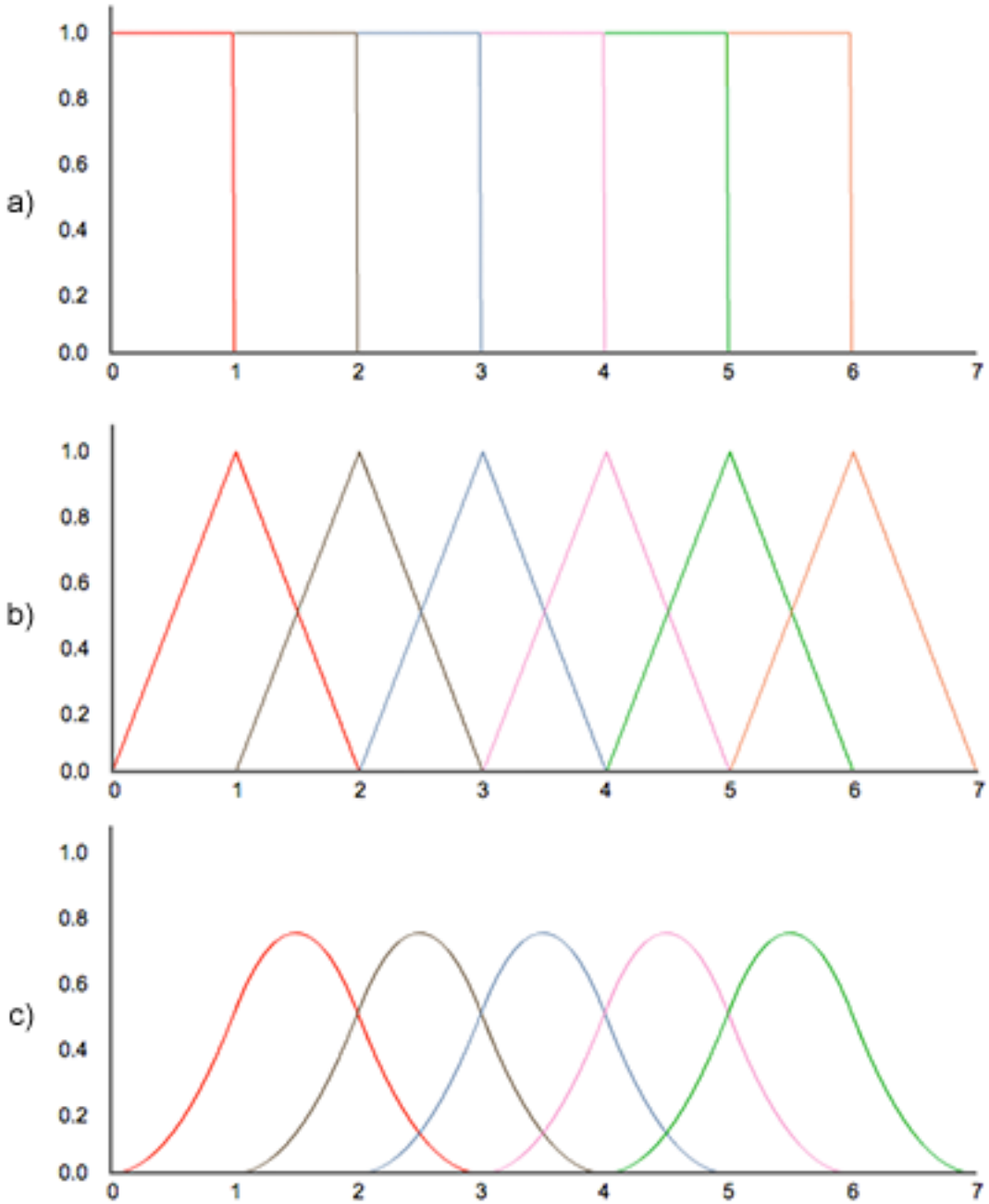
Všechny tyto i jiné tvary křivek je možné modelovat pomocí appletu, který je součástí této diplomové práce.

3.2 B-spline bázové funkce

Podobně jako u Bézierových křivek používáme pro jednu z možností jejich výpočtu Bernsteinovy polynomy, u NURBS křivek máme B-spline bázové funkce. Opět jsou v podstatě zobecněnými Bernsteinovými polynomy, neboť pro body P_i s váhami

w_i rovny jedné a uzlový vektor (3.6) jsou B-spline křivky rovny Bernsteinovým polynomům.

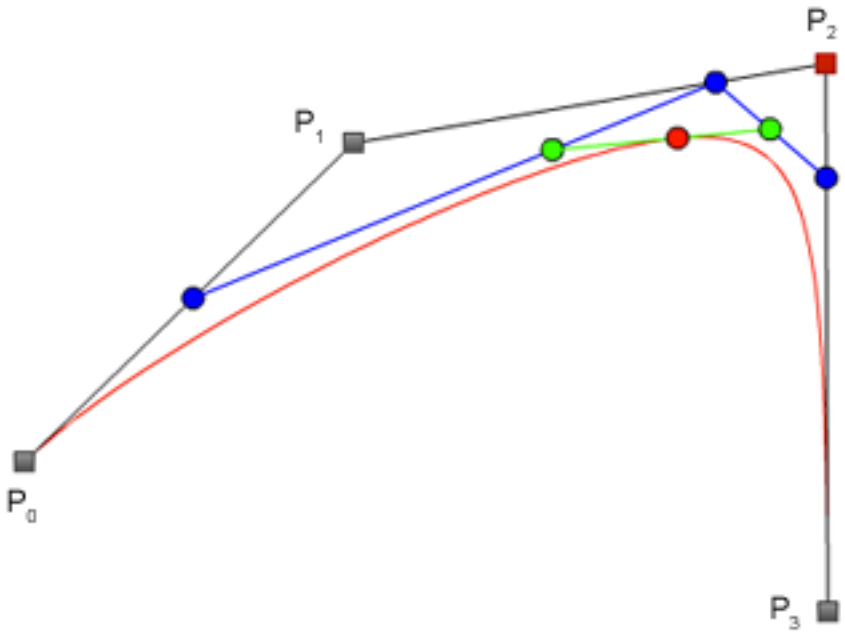
Výpočet i-té B-spline křivky k-tého řádu získáme pomocí rekurzivního výpočtu podle rovnice (3.3). Na obrázku 3.4 níže můžeme vidět B-spline bázové funkce pro $i = 0, 1, \ldots, 4, 5$ a pro $n = 0, 1, 2$, tj. $k = 1, 2, 3$ vygenerované appletem pro uzlový vektor $U = \{0, 1, 2, 3, 4, 5, 6, 7\}$.



Obr. 3.4: B-spline bázové funkce a) 0-tého řádu b) 1-ního řádu c) 2-hého řádu.

3.3 De Boorův algoritmus

Stejně jako algoritmus de Casteljau patří mezi základní výpočetní techniky pro Bézierovu křivky, tak i de Boorův algoritmus nám určuje bod na křivce v čase t. V podstatě je de Boorův algoritmus zobecněním algoritmu de Casteljau pro B-spline křivky s neuniformní parametrizací. To znamená, že se de Boorův algoritmus od algoritmu de Casteljau liší tím, že nepoužívá při konstrukci konstantní rozdělení stran řídícího polygonu.



Obr. 3.5: De Boorův algoritmus

Hledání bodu na křivce probíhá v několika krocích, nejprve hledáme část řídícího polygonu, která ovlivňuje segment křivky, jemuž bod náleží. Po nalezení příslušného intervalu v uzlovém vektoru je aplikováno dělení, které používá neuniformní časové intervaly určené uzlovým vektorem. [1]

Vstup: čas $t \in \langle 0, 1\rangle$, Uzlový vektor $\{u_i\}_{i=0}^{n+k+1}$, řídící body $\{P_i\}_{i=0}^{n}$ a váhový vektor $\{w_i\}_{i=0}^{n}$.

1. vytvoříme pole bodů $p[0][i][j] = P_i, i = 0, \ldots, n$ a $s = 0, \ldots, 3$.
2. do posledního vnořeného pole $[s]$ uložíme jednotlivé souřadnice bodu vynásobené příslušnou váhou, tj. $[0] = P_i x w_i, [1] = P_i y w_i$ a jako poslední složku bodu uložíme jeho váhu $[2] = w_i$.
3. Pro dané $t \geq t_k$ nalezneme interval $t \in \langle u_j, u_{j+1})$ z uzlového vektoru U.
4. Pro $p = 1, \ldots, k$ opakuj.
   Pro $i = j - k + p, \ldots, j$ opakuj.
   $P[m][i] = \frac{t - t_i}{t_{i+k-(p-1)} - t_i} P[m-1][i] + \frac{t_{i+k-(p-1)} - t}{t_{i+k-(p-1)} - t_i} P[m-1][i-1]$.
5. $P(t) = P[j][k]$.

De Boorův algoritmus je velmi rychlý a numericky stabilní algoritmus, který vymyslel Carl R. de Boor. Později byl algoritmus zjednodušen jako jeho rychlejší varianta, ale tento modifikovaný algoritmus ztrácí stabilitu.

Na obrázku 3.5 je vidět NURBS křivka vykreslena pomocí tohoto algoritmu s konstrukcí bodu v čase $t = 0.6$. Uzlový vektor je $U = \{0, 0, 0, 0, 1, 1, 1, 1\}$ a váhy jsou $W = \{3, 7; 2.6, 5.7, 1\}$.

4 RASTERIZACE

V počítačové grafice se s pojmem rasterizace setkáváme, pokud mluvíme o způsobu zobrazení grafických prvků jako jsou např. úsečky, lomené čáry, kružnice nebo elipsy. Tyto prvky můžeme zobrazit jako posloupnost pixelů nebo posloupnost úseček. V prvním případě se jedná o rastrové vykreslení a ve druhém případě jde o vykreslení vektorové. Současná počítačová grafika se soustřeďuje především na rasterovou grafiku, proto je potřeba převádět vektorový výstup do rasterové podoby. Tento proces se nazývá rasterizací. Jinými slovy lze popsat rasterizaci jako výběr pozice a barvy pixelů tak, aby co nejvěrněji připomínaly tvar, barvu a polohu zadaného vektorového grafického prvku. [1]

Mezi nejjednodušší a zároveň nejdůležitější grafický prvek patří úsečka. Pomocí úseček různé délky jsme schopni na obrazovce výstupního zařízení vykreslit prakticky jakýkoli tvar či grafický prvek. Nejkratší úsečku, kterou na daném zařízení můžeme zobrazit, je úsečka o délce rovné velikosti nejmenšího bodu, který lze na zařízení zobrazit. Úsečka je zadávána pomocí souřadnic dvou bodů $[x_1, y_1]$ a $[x_2, y_2]$, přičemž je dobré zachovávat konvenci kreslení úsečky z počátečního bodu $[x_1, y_1]$ do koncového bodu $[x_2, y_2]$. U úsečky nás dále zajímá její směrnice m, která je definována následovně: [1]

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \tag{4.1}$$

Velikost absolutní hodnoty $|m|$ určuje, která osa bude řídící a která vedlejší. Řídící osou je vždy ta, v jejímž směru se vzorkuje. Pro $|m| < 1$ je řídící osou osa x, pro $|m| > 1$ osa y. Diagonální úsečky ($|m| = 1$) mohou mít řídící osu libovolnou.[1]

Mezi algoritmy zabývajícími se problémem rasterizace patří DDA algoritmus a Bresenhamův algoritmus. Vykreslením tlusté čáry se pak zabývá Murphyho algoritmus, který je rozšířením Bresenhamova algoritmu.

Součástí této diplomové práce je návrh a realizace právě Bresenhamova algoritmu, který se často používá v počítačové grafice pro snadnost jeho výpočtu.

4.1 DDA algoritmus

Patří mezi první a nejznámější algoritmy pro rasterizaci. Je založen na postupném přičítání konstantních přírustků k oběma souřadnicím, počínaje prvním bodem úsečky. Výpočet aktuální souřadnice bodu se provede přičtením přírůstku k předchozím hodnotám souřadnic. [1]

$$x = x_1 + t\Delta x \tag{4.2}$$
$$y = y_1 + t\Delta y \tag{4.3}$$

Ve směru řídící osy se přičítá konstanta 1, na vedlejší ose konstanta m v případě osy x

$$x_{k+1} = x_k + 1 \tag{4.4}$$
$$y_{k+1} = y_k + m, \tag{4.5}$$

pro $k = 1, 2, 3, \ldots, n$, repektive $1/m$ v případě řídící osy y. Po přičtení konstant je potřeba souřadnice zaokrouhlit na celá čísla.[1]

Pro $|m| < 1$ bude postup následující:

1. Ze zadaných bodů $[x_1, y_1]$ a $[x_2, y_2]$ vypočteme směrnici m podle (4.1).
2. Inicializujeme aktuální bod $[x, y]$ hodnotou počátečního bodu $[x_1, y_1]$.
3. Dokud je $x \leq x_2$, opakujeme:
   - Vykreslíme zaokrouhlené souřadnice $[x, y]$
   - $x = x + 1$
   - $y = y + m$

4.2 Bresenhamův algoritmus

Při rasterizaci rozhodujeme, který pixel nejblíže ležící skutečné úsečce vybereme, abychom dostali co možná nejlepší výsledek. Protože nejmenším bodem v počítačové grafice na daném zobrazovacím zařízení je pixel, výstupem z algoritmu by měly být celočíselné souřadnice vybraných pixelů. J. E. Bresenham vyvinul velmi efektivní algoritmus, který má oproti DDA algoritmu výhodu použití celočíselné aritmetiky. [1]

Předpokládejme, že je již nakreslen bod o souřadnicích $[x_i, y_i]$. V dalším kroku $n+1$ máme na výběr ze dvou souřadnic, a to $[x_i + 1, y_i]$ a $[x_i + 1, y_i + 1]$. Vzdálenosti souřadnic y středů těchto dvou pixelů jsou označeny jako d_1 a d_2. Jejich hodnoty vypočítáme, vyjdeme-li z obecné rovnice přímky [1]

$$y = mx + b, \tag{4.6}$$

kde m je směrnice a b posun na ose y. Rozdílem vzáleností d_1 a d_2, získáme rozhodovací prvek algoritmu Δd:

$$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1} \tag{4.7}$$

Definujeme si dvě proměnné d_1 a d_2, což jsou vzdálenosti skutečného bodu od souřadnic y_i a $y_i + 1$

$$d_1 = y - y_i \tag{4.8}$$
$$d_2 = y_i + 1 - y. \tag{4.9}$$

Rozdíl těchto dvou vzdáleností vyjádříme jako

$$\Delta d = d_1 - d_2. \tag{4.10}$$

Nyní jsme schopni rozhodnout, který z možných pixelů leží blíže y-ové součadnici skutečnému bodu. Nezáleží nám na velikosti proměnné Δd, ale na jejím znaménku. Je-li hodnota Δd záporná, leží blíže pixel se souřadnicí y_i, kladná hodnota značí opak, tedy blíže leží pixel o souřadnici $y_i + 1$. Tento jev je výhodný pro převod výpočtů do celočíselné aritmetiky. Proto si definujeme proměnnou p_i [1]

$$p_i = \Delta d \Delta x = 2\Delta y - x_i, \tag{4.11}$$

kterou můžeme označit jako rozhodovací člen. Algoritmus pro řídící osu x postupuje takto

1. Určíme si konstanty k_1 a k_2 z koncových bodů
   - $k_1 = 2\Delta y$
   - $k_2 = 2(\Delta y - \Delta x)$
2. Inicializujeme rozhodovací člen p na hodnotu $p = 2\Delta y - \Delta x$
3. Inicializujeme $[x, y]$ jako $[x_1, y_1]$
4. Vykreslíme bod $[x, y]$
5. Dokud je $x \leq x_2$, opakujeme:
   - $x = x + 1$
   - Je-li $p \leq 0$, pak $y = y + 1$ a $p = p + k_2$
   - Je-li $p < 0$, pak $p = p + k_1$
   - Vykreslíme bod $[x, y]$

V každém kroku na základě znaménka rozhodovacího členu p_i vypočítáme hodnotu p_{i+1} přičtením konstanty k_1 nebo k_2 a pokud je p_i kladné, iterujeme hodnotu

y na $y = y + 1$. Jedna z konstant je záporná a druhá kladná. Tím se docílí změna znaménka rozhodovacího členu p_i. [1]

Následující příklad demonstruje rasterizaci úsečky pomocí Bresenhamova algoritmu pro řídící osu x, tedy pro úsečky s úhlem $\alpha \in \langle 0°, 45° \rangle$. Pokud bychom chtěli kreslit všemi směry musíme změnit řídící osu, případně překlopit křivku do příslušného kvadrantu. [1]

4.3 Rasterizace silné čáry

V roce 1987 Alan Murphy s podporou IBM publikoval tento modifikovaný algoritmus pro kresbu tlusté čáry (thick line). Je založen na algoritmu J. E. Bresenhama. Vlastnictví práv algoritmu náleží společnosti IBM (copyright IBM Corporation 1978). [10]

Pokud chceme vykreslit silnou čáru, můžeme použít rychlý postup, který je nepřesný, nebo výpočetně složitější postup, který tyto nepřesnosti řeší. Jednodušší algoritmy, mezi které patří Murphyho modifikovaný Bresenhamův algoritmus, vytvářejí silnou čáru přidáním jednoho či několika pixelů nad sebou či vedle sebe. Tato metoda má dva negativní jevy. [1]

1. Síla čáry se mění podle sklonu úsečky.
2. Zakončení čar je nepřirozené a rovnoběžné s jednou ze souřadnicových os.

Ilustraci prvního problému můžeme vidět na obrázku 4.1, kde tloušťka šikmé čáry je menší než tloušťka vodorovné či svislé čáry. Tento jev lze částečně odstranit u silnějších čar, kde můžeme tloušťku korigovat podle toho, zda je čára více rovnoběžná s některou ze souřadnicových os nebo zda je šikmá. [1]

Sofistikovanější algoritmy pracují s tlustou čarou jako s vyplněnou plochou. [1]

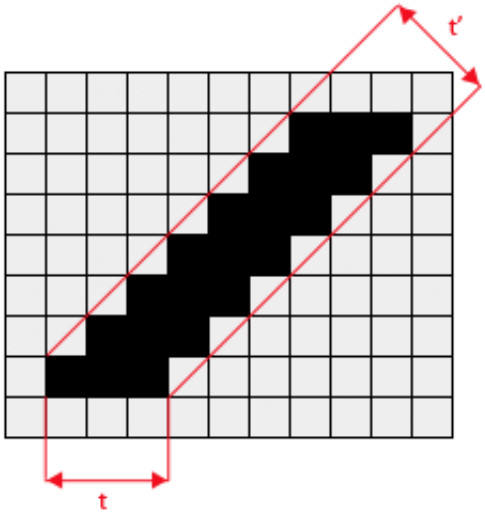
4.4 Rasterizace kružnice a elipsy

J. E. Bresenham publikoval algoritmus pro rasterizaci kružnice, který je stejně efektivní jako jeho vyřešení rasterizace úsečky. V literatuře je algoritmus nazván „midpoint algorithm“. [1] Pro body na kružnici platí rovnice $x^2 + y^2 - r^2 = 0$, kterou lze zapsat v podobě funkce [1]

$$F(x, y) : x^2 + y^2 - r^2 = 0. \tag{4.12}$$

Hodnota funkce F je pro body ležící uvnitř kružnice záporná, zatímco pro body ležící vně kružnice je hodnota kladná. [1]

Ukažme si situaci, kdy se nacházíme na pozici $[x_i, y_i]$ (na obrázku 4.2 vyznačeno modře). Dalším bodem, který vybereme může být buď bod $[x_i + 1, y_i]$ nebo



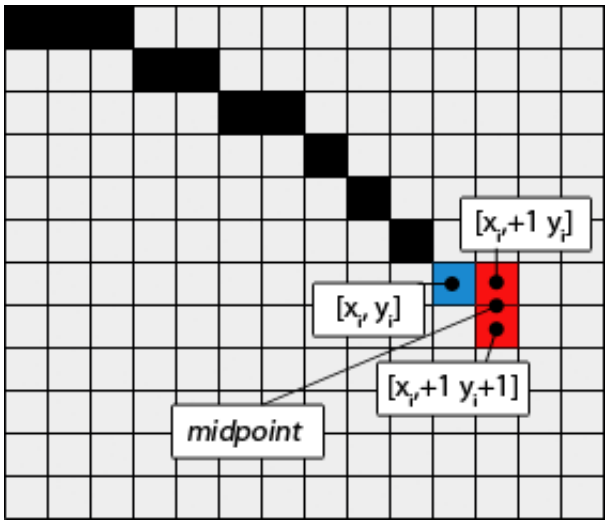
Obr. 4.1: Proměnná tloušťka čáry související se směrnicí čáry $t \neq t'$

$[x_i + 1, y_i + 1]$ (na obrázku 4.2 jsou oba body vyznačeny červeně). Určíme si bod, ležící na středu úsečky mezi těmito dvěma body. Tento bod je rovněž vyznačen na obrázku a je označen jako „midpoint“. Dosadíme-li jeho souřadnice $[x_i + 1, y_i - 1/2]$ do funkce (4.12), rozhodneme na základě znaménka výsledku o tom, zda tento bod leží uvnitř či vně kružnice. Výsledek tedy opět můžeme označit jako rozhodovací činitel p_i [1]

$$p_i = F\left(x_i + 1, y_i - \frac{1}{2}\right) = (x_i + 1)^2 + \left(y_i - \frac{1}{2}\right)^2 - r^2. \tag{4.13}$$

Je-li znaménko p_i záporné, bude vybrán bod se stejnou souřadnicí y_i, tedy ten „horní“. Naopak při kladné hodnotě rozhodovacího členu p_i bude vybrán pro vykreslení bod $y_i + 1$, tedy ten „spodní“. Hodnotu p_i budeme pro každý krok algoritmu přepočítávat podle předchozí hodnoty. [1] Tedy takto

$$p_{i+1} = p_i + 2x_i + 3 - \left(y_i - \frac{1}{2}\right)^2 + \left(y_{i+1} - \frac{1}{2}\right)^2. \tag{4.14}$$



Obr. 4.2: Ukázka rasterizace kružnice

1. Inicializujeme pomocné proměnné $dvex = 3$, $dvey = 2r - 2$
2. Inicializujeme rozhodovací člen $p_i = 1 - r$
3. Inicializujeme $[x, y]$, jako $[0, r]$
4. Vykreslíme bod $[x, y]$
5. Dokud je $x \leq y$, opakujeme:
   - Vykreslíme 8 bodů symetrickým s bodem $[x, y]$
   - Je-li $p > 0$, pak
     - $p = p - dvey$
     - $dvey = dvey - 2$
     - $y = y - 1$
   - $p = p + dvex$
   - $dvex = dvey + 2$
   - $x = x + 1$

Rasterizace elipsy je méně snadný úkol. Narozdíl od kružnice lze symetrii použít pouze pro tři body. Rovnici elipsy se středem v počátku vyjádříme pomocí funkce [1]

$$F(x, y) : b^2x^2 + a^2y^2 - a^2b^2 = 0. \tag{4.15}$$

Z této rovnice opět odvodíme pomocí středového bodu rozhodovací člen p_i, obdobně jako u kružnice. [1]

Protože nemůžeme využít symetrie jako u kružnice, budeme muset měnit řídící osu v průběhu vykreslování. Generujeme-li body v prvním kvadrantu zleva doprava,

bude řídící osou nejprve osa x, poté osa y. Bod, ve kterém dojde ke změně řídící osy má tečnu se směrnicí $m = -1$ k elipse. Souřadnice bodu lze spočítat následovně

$$\left[\frac{a^2}{\sqrt{a^2+b^2}}, \frac{b^2}{\sqrt{a^2+b^2}}\right]. \tag{4.16}$$

Pro řídící osu x bude výpočet rozhodovacího členu probíhat takto

$$p_i = \begin{cases} p_{i+1} = p_i + b^2(2x_i + 1) & \text{pro } p_i \leq 0 \\ p_{i+1} = p_i + b^2(2x_i + 1) - 2a^2 y_i & \text{pro } p_i > 0 \end{cases}, \tag{4.17}$$

pro osu y budou vztahy obdobné.

5 PRAKTICKÁ REALIZACE

5.1 RIA (Rich Internet Application)

Za posledních několik let zažil svět world wide webu ohromný rozvoj. Stále více lidí používá internet k hledání informací, k zábavě, ke komunikaci s ostatními lidmi apod. Dnešní trend došel tak daleko, že v internetových sítích vytváříme nové světy v podobě sociálních sítí jako je Facebook, sdílíme informace pomocí Twitter, nakupujeme v internetových obchodech a používáme aplikace jako jsou například Google Docs. Kam budeme směřovat v budoucnu, se teprve uvidí. Ale už nyní je jasné, že nás čekají velké věci. Stejně jako před deseti lety byly projekty typu Wikipedia či YouTube utopií, v budoucnu se rovněž mohou realizovat projekty, které nám dnes přijdou nepředstavitelné.

Už dnes je však jasné, že klasické standartní technologie typu HTML verze 4 nebudou stačit k naplnění budoucích vizí. Upřímně ony už nestačí ani dnes. To je důvod proč již několik let zpátky vznikl nápad vytvořit prostředí pro bohaté internetové aplikace. Poprvé se tento termín představen v roce 2002 společností Macromedia, která je nyní sloučená se společností Adobe. Tento nápad již dříve pod jiným názvem mělo i několik dalších firem a vizionářů. Jednou z firem byl i Microsoft, který tento nápad nazval Remote Scripting již v roce 1999. [6]

Pokud tedy máme mluvit o „bohatých internetových aplikacích“, musíme si je nejdříve charakterizovat. RIA je internetová aplikace, která by měla být plynulým nástupcem současných webových technologií, které jsou v mnoha směrech velmi omezené. Pokud chceme v současné době realizovat koncept RIA aplikace, máme dvě možnosti. Buď použijeme některou z technologií, která vyžaduje nainstalovaný zásuvný modul ve webovém prohlížeči (Adobe Flex, Java, Microsoft Silverlight) nebo využijeme JavaScriptovou alternativu, jakou je Ajax či plánované HTML 5.

U první skupiny máme velkou výhodou, že lze pomocí několika kliknutí nainstalovat správnou verzi zásuvného modulu pro danou aplikaci, aniž bychom problematice museli rozumět. Zároveň máme jistotu, že se webová aplikace zobrazí a bude fungovat tak, jak má, na všech zobrazovacích zařízeních, což se nedá s jistotou říci u druhé skupiny technologií. I přesto zůstávají technologie jako je Flex, Java, Silverlight alternativou bohatých aplikací. Proto si představme nejprve technologie Ajax a HTML 5. Začneme Ajaxem, neboť jde o technologii aktuálně použitelnou, zatímco HTML 5 je pouze vyvíjenou rozšířenou specifikací.

5.1.1 Ajax

Ajax je zkratkou pro Asynchronní JavaScript a XML. Můžeme tedy posílat nebo dostávat data na server a ze serveru asynchronně, tedy v pozadí běhu internetové aplikace. Příkladem může být situace, kdy návštěvník webu klikne na některý odkaz a my, jako autoři, nechceme znovu přenášet celou aplikaci jen kvůli změně obsahu v některé části webu. V takovém případě použijeme Ajax, který pošle dotaz na server a vzápětí dostane odpoveď pouze s požadovaným obsahem, který následně zobrazí, aniž by musel znovu načíst již načtené položky jako je menu, logo apod. Celá aplikace reaguje rychleji a odpadají nepříjemnosti s přebliknutím stránky při znovunačtení. Ajax pro komunikaci se serverem využívá objekt XMLHttpRequest za použití XML nebo JSON standartu. Hlavní problém Ajaxu je, že je v podstatě absolutně závislý na prohlížeči, ve kterém internetová aplikace běží. I když moderní prohlížeče již Ajax plně podporují, stačí mít z nějakého důvodu JavaScript v prohlížeči vypnutý a web nebude správně pracovat. Jedna z jeho dalších nevýhod je, že kód celé aplikace je přístupný prakticky komukoli a neexistuje způsob, jak toto duševní vlastnictví utajit. Nicméně jde o velmi přijatelnou technologii, která v mnohém napomůže zlepšit a oživit celou webovou aplikaci. [7]

5.1.2 HTML 5

HTML 5 je rozšiřující specifikace známého značkovacího jazyka HTML. V současnosti je celá specifikace ve stádiu návrhu organizací W3C. Tato specifikace by měla přinést nové značky sémantického významu, podporu offline prohlížení webu, zjednodušenou syntax a spouštění multimediálního obsahu. Je to nepochybně krok vpřed ve vývoji RIA aplikací, nicméně není jisté, zda se podaří specifikaci HTML 5 vytvořit tak, aby se zajistilo vývojářům správné zobrazování jejich aplikací ve všech zobrazovacích zařízeních světa. Není žádným tajemstvím, že toto je velkým každodenním problémem webových vývojářů. V podstatě jde o to, že každý prohlížeč si kód webu přeloží, jak uzná za vhodné, a zobrazí dle svých standartů, které se bohužel u různých prohlížečů liší. Mozzila Firefox je v současné době prohlížečem, který je nejblíže k podpoře standartů vydané organizací W3C. Zdánlivé jednoduchosti, jakou je například zaoblení rohů, se pak stávají noční můrou, se kterou se nelehko vypořádává lecjaký vývojář. Uvidíme, co nám specifikace HTML 5 přinese. [8]

5.1.3 Silverlight

Mezi tuto skupinu aplikací patří vývojové prostředí Microsoft Silverlight. Je založeno na Windows Presentation Foundation (WPF), což je poslední vyvinutá UI technologie .NET frameworku. Uživatelská rozhraní a layout jsou vyvíjena v jazyku XAML,

založeném na XML značkovacím jazyku. Jazyky C# nebo VB.NET se používají k vyvíjení logických částí kódu. Hlavní výhodou Silverlight je podobnost s ostatními .NET prostředími, což je příjemné pro mnoho .NET vývojářů. Nevýhodou je nižší procento uživatelů, kteří mají nainstalován zásuvný modul pro běh Silverlight v porovnání s technologií Flex. Je zde tedy větší pravděpodobnost, že se webový obsah nezobrazí vůbec, než v případě Flexu. [11]

5.2 Flex

Nyní popíšeme detailněji vývojové prostředí Adobe Flex, které používáme pro vývoj praktické části diplomové práce, k vývoji interaktivních appletů pro výuku počítačové grafiky.

Flex je souborem vývojových technologií, který umožňuje vývojářům velmi rychle vytvářet aplikace, které mohou fungovat jako desktopové aplikace stejně dobře jako applety ve webovém prohlížeči. Flex je možné chápat i jako framework, nebo-li soubor komponent, tříd, kompilátorů a dalších nástrojů, za jejich pomoci se vývojář může soustředit na skutečný vývoj místo řešení problémů na nižších programovacích úrovních. Flex používá dvě běhové prostředí, a to Adobe Flash Player, čímž je zaručená funkčnost v 99 procentech webových prohlížečů, a Adobe Air, díky kterému jsme schopni vytvářet samostatné (stand alone) aplikace běžící multiplatformě na většině běžně užívaných operačních systémech včetně Microsoft Windows, Mac OS, různé distribuce Linux apod. Flex také umožnuje funkčnost aplikací na mobilních zařízeních jakou jsou tablety či chytré telefony. [9]

Flex využívá existujících a funkčních technologií jako jsou například XML, webové služby, HTTP protokolu a Action Scriptu. Také Flex umožňuje vytvářet celistvé bohaté internetové a desktopové aplikace, a to jednoduchým a intuitivním způsobem. I přes jeho relativní jednoduchost je Flex silným objektově orientovaným nástrojem, který umožňuje vývojářům převzít kontrolu nad nižšími programovacími úrovněmi, vytvářet vlastní třídy, komponenty, vytvářet layouty, datové modely apod.[2]

5.3 Flex není Flash

Technologie Flash, původně od firmy Macromedia, se proslavila přínosem doposud neuskutečnitelných možností a posunula směr a vývoj internetu. Internet nebo-li World Wide Web byl navržen především jako zdroj informací v textové podobě. V prvních webových aplikacích se nevyskytovaly obrázky či jiná média. Šlo pouze

o prezentaci textu, který byl strukturován pomocí značkovacího jazyka HTML. Později ale začala narůstat potřeba umístit na web obrázky, animované obrázky (nejprve v podobě souborů .gif), zvuk (wav a mp3 formát) a později i video. V klasickém značkovacím jazyku HTML nelze prozatím nadefinovat video interpreter. Z tohoto důvodu se na to musí jinak. Jedna z cest byl a je Flash. [12]

Flash přináší do webového světa možnost umístit na web multimediální obsah v podobě tzv. appletů. I přes tyto kvality je Flash založen na poměrně jednoduché a již zastaralé koncepci klíčových snímků, přechodů a časové osy. Je proto vhodný na animace, bannery a jiné grafické aplikace, nikoli však jako alternativa současně se rozvíjejícího webu. To je také nejspíš jeden z důvodů, proč v budoucnu nejspíše vymizí úplně. [12]

Flex narozdíl od Flashe je programátorská záležitost, není založen na konceptu časové osy a v současné době patří mezi nejlepší vývojové technologie. S Flashem má společného snad jen použití Adobe Flash playeru, který používá jako interpreter. [12]

5.4 Jazyk MXML

Jazyk MXML je značkovací jazyk založený na XML koncepci. Pomocí definovaných značek jsme schopni strukturovat grafické rozložení a design aplikace. To ovšem není vše. Pomocí MXML jsme schopni vkládat tlačítka, formuláře, obrázky, různé grafické prvky, multimedia, animace, efekty a datové modely. MXML je dostatečně robustní jazyk na to, abychom vytvořili celé aplikace čistě jen v tomto jazyku. [2]

MXML dále přináší WYSIWYG rozšíření, a tím dává možnost vytvářet aplikace bez znalosti jakékoli syntaxe. V praxi je ovšem většinou tato možnost využita k dosažení rychlejšího výsledku či možnosti lepšího grafického vnímání pro umístění komponent.

Dalo by se říci, že jazyk MXML je příjemným a intuitivním prostředím, které je postaveno nad objektově orientovaný jazyk ActionScript. Jinými slovy při kompilaci kódu přeloží kompilátor kód MXML na kód ActionScriptový. I přesto je pro složitější a sofistikovanější aplikace zapotřebí znalost jak jazyka MXML, tak i ActionScriptu. To znamená, že MXML dokument nám dává sílu objektově orientovaného jazyka spolu s přívětivým přístupem značkovacího jazyka. [2]

Jako malý příklad rozebereme následující MXML kód:

Příklad kódu 5.1: Aplikace pro přihlášení v MXML kódu.

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                       xmlns:s="library://ns.adobe.com/flex/spark"
                       xmlns:mx="library://ns.adobe.com/flex/mx">
```

```
        <mx:Form id="myForm" defaultButton="{prihlasit}">

                <mx:FormItem label="uživatelské jméno">
                        <mx:TextInput name="uzivatelske_jmeno"/>
                </mx:FormItem>
                <mx:FormItem label="heslo">
                        <mx:TextInput name="heslo"/>
                </mx:FormItem>
                <mx:FormItem>
                        <mx:Button label="přihlásit" id="prihlasit"/>
                </mx:FormItem>

        </mx:Form>

</s:WindowedApplication>
```

První řádek je XML deklarace. Deklaruje použitou verzi a kódování. Párová značka <s:WindowedApplication> poukazuje na to, že naše aplikace je desktopová. Zároveň se v ní deklarují jmenné prostory, na základě kterých se načítají příslušné knihovny. Dále již následuje kód popisující rozložení komponent nebo-li layout. Toto téma bude detailněji rozebráno v následující kapitole.

Kód je velmi jednoduchý, vytváří formulář pro přihlášení uživatele. Prozatím zde není naprogramována žádná funkčnost. Komponenty Form, FormItem, TextInput, Button jsou ActionScriptové třídy, které jsou součástí MX balíčku ve Flex SDK. Stejně jako mnoho dalších komponent ani tyto nemusíme vytvářet, jsou již hotovy a připraveny pro naše libovolné použití. Právě v tom se skrývá síla Flexu. Výsledek kompilace tohoto kódu je na obrázku 5.1.

Obr. 5.1: Aplikace pro přihlášení.

5.5 Layout

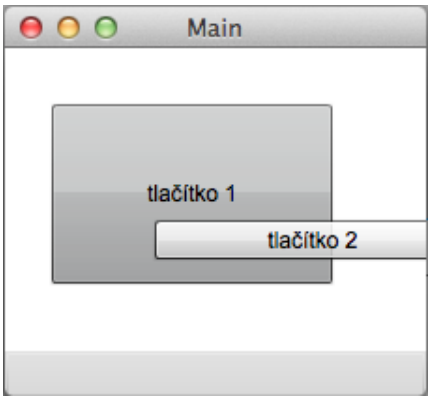
Layout je v kontextu k jazyku Flex chápán jako rozmístění komponent dané aplikace. Jedna z klíčových výhod Flexu oproti jiným technologiím je, že ve Flexu lze pracovat

s mnoha layout kontainery, které za nás řeší spoustu problémů. Jsou jimi např. kontainery Group, VGroup, HGroup apod. [2]

Obecně bychom mohli rozdělit rozmístění komponent na absolutní rozmístění a na relativní. Rozdíl je zřejmý. Absolutně rozmístěné prvky mají přesně danou svou pozici, která se váže k souřadnici $[0, 0]$, umístěné v levém horním rohu aplikace. Zatímco u relativního rozmístění komponent bereme v úvahu velikost okna aplikace, vztahy mezi jednotlivými komponentami apod. Při každé změně některého ze závislých vztahů se celý layout aplikace překreslí s novým rozmístěním. Jako příklad uveďme zmenšení velikosti okna aplikace. Při takové změně se všechny relativně rozmístěné prvky znovu rozmístí tak, aby jejich pozice co nejlépe vyhovovala dané situaci, zatímco absolutně umístěné komponenty zůstanou na stejném místě.

Příklad kódu 5.2: Příklad absolutního layoutu

```
<s:Button x="11" y="7" width="150" height="96" label="tlačítko 1"/>
<s:Button x="33" y="70" width="169" label="tlačítko 2"/>
```



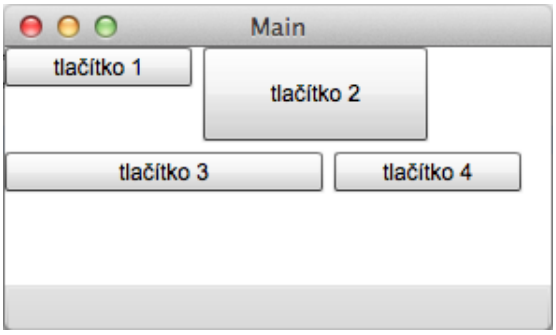
Obr. 5.2: Příklad absolutního layoutu podle kódu 5.2

Pozice prvku je nastavena pomocí vlastností x a y. Pokud se prvky překrývají podobně jako v našem případě, vykreslí se v pořadí, v jakém jsou uvedeny v kódu. Tedy tlačítko 2 bude překrývat tlačítko 1. Na obrázku 5.2 je možné vidět, že kdyby se obě tlačítka posunula do levého horního rohu, aktuální velikost okna by stačila pro jejich vykreslení. Protože jsou ale absolutně rozmístěné, zůstanou na své pozici.

Příklad kódu 5.3: Příklad relativního layoutu

```
<s:VGroup>
        <s:HGroup>
                <s:Button width="100" label="tlačítko 1"/>
                <s:Button width="120" height="50" label="tlačítko 2"/>
        </s:HGroup>
```

```
        <s:HGroup>
                <s:Button width="170" label="tlačítko 3"/>
                <s:Button width="100" label="tlačítko 4"/>
        </s:HGroup>
</s:VGroup>
```



Obr. 5.3: Příklad relativního layoutu podle kódu 5.3

U jednotlivých komponent lze velmi snadno nastavovat okraje a odsazení.

5.6 Adobe Air

Adobe Air je prostředí, které umožňuje vytvářet samostatné desktopové aplikace, běžící na všech běžných platformách a zařízeních včetně osobních počítačů s operačními systémy MacOS, MS Windows, Linux, také na mobilních zařízeních s operačními systémy Android™, BlackBerry®, iOS a dokonce na televizích. [9]

S Adobe Air můžeme vyvíjet internetové aplikace pomocí HTML, JavaScript, ActionScript, Flex a Adobe Flash, které se chovají jako programy bez podpory prohlížeče. Zatímco aplikace spouštěné ve webových prohlíženích nepotřebují instalovat, Adobe Air aplikace instalaci potřebují. S tím je svázáno zabalení aplikace, digitální podpis, nicméně díky tomu mohou aplikace přistupovat na disk a chovat se jako běžné aplikace. Zatímco aplikace spuštěné prohlížečem jsou velmi omezené. [5]

Adobe Air podporuje SQLite databázi, což je pro vývojáře velmi příjemná záležitost. SQLite simuluje databázový server, což umožňuje přístup přes SQL syntax, ale data ukládá do souboru. I přes některé nevýhody se vývojář vyhne spoustě nepříjemnostem, které se váží k ukládaní dat pomocí souboru. [5]

6 VÝSLEDKY STUDENTSKÉ PRÁCE

6.1 Řešení appletů

Applety jsou vytvořeny pomocí technologie Flex a spouští se přes běhové prostředí Flash Player a Adobe AIR, který je zdarma ke stáhnutí na webu Adobe `http://www.adobe.com` pod položkou „Downloads“ v horním menu. Pro spuštění appletů tedy potřebujeme pouze webový prohlížeč a nainstalovaný Flash Player, standalone aplikace již mají běhové prostředí v sobě.

Pro praktickou realizaci appletů jsem naprogramoval ActionScriptové třídy BresenhamAlgorithm.as, MidpointAlgorithm.as, Nurbs.as, Bezier.as, RationalBezier.as a BSpline.as. Všechny tyto třídy jsou logické, tj. jejich úkolem je vypočítat výstupní data na základě dat vstupních. Je možné je nasadit na jakýkoli grafický interface. Všechny applety mají jednotný vzhled a podobné, pokud ne stejné ovládání. V rámci jejich GUI jsem vytvořil třídu Graph.as, která dědí ze třídy SkinnableContainer, a příslušné skiny, které efektivně řeší vzhled celé aplikace. Všechny kódy se nacházejí buď v příloze, na CD nebo na webu. Je možné stáhnout také přímo vytvořené projekty pro Flash Builder. Projekty a jejich kódy jsou vytvořeny v programu Flash Builder pro Flex SDK 4.6 (v době psaní této práce nejaktuálnější verze).

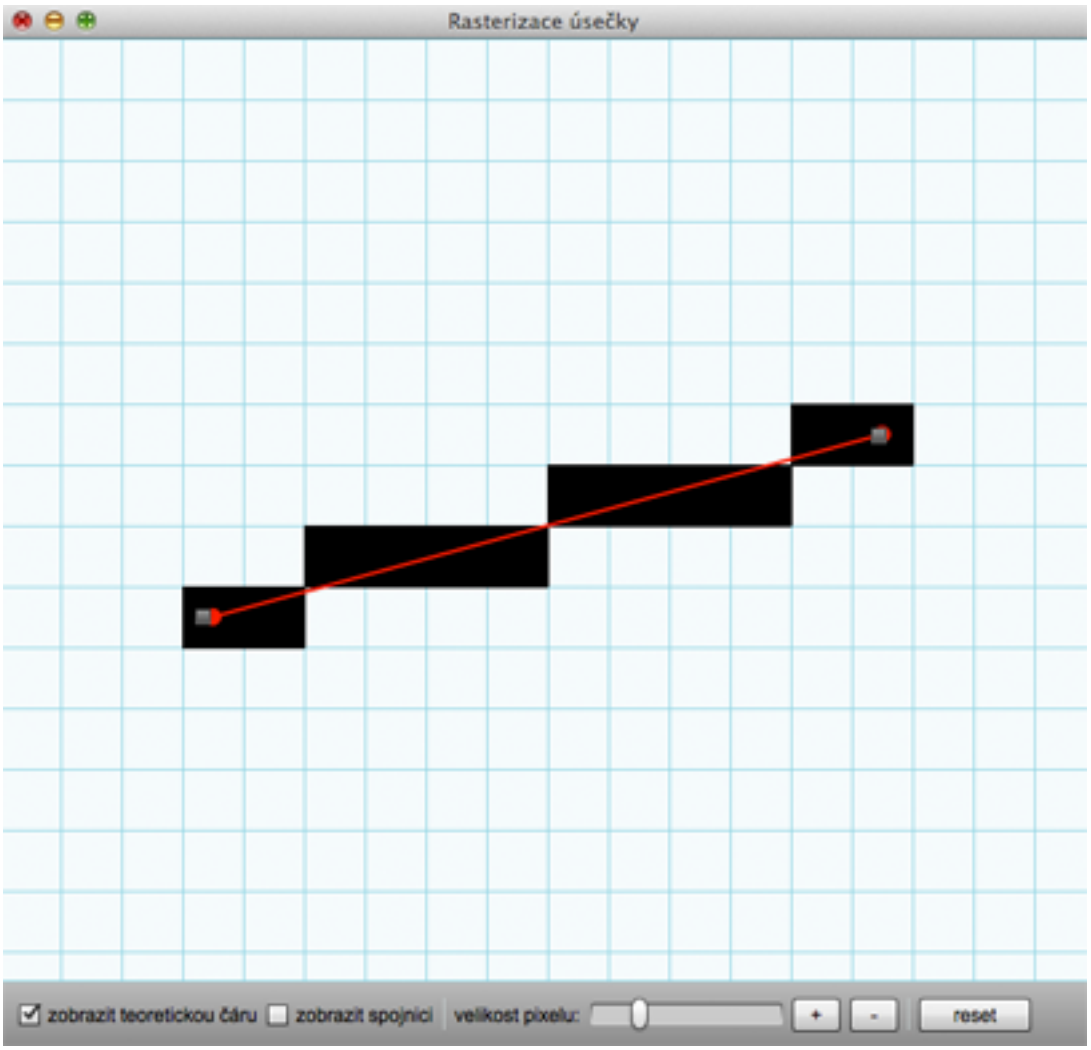
6.2 Ovládání appletů

Applety mají velmi jednoduché a intuitivní ovládání. Každý se skládá z vykreslovací plochy a ovládacího panelu dole, některé jako například applet de Boorova algoritmu mají navíc ještě panel se specifickým nastavením. Většina appletů umožňuje maximalizaci na celou obrazovku tak, aby se zvětšila vykreslovací plocha. Individuální návod pro každý applet je popsán v této části diplomové práce, vždy pod daným appletem a také na webových stránkách, které jsou součástí praktické části práce.

6.3 Applet pro rasterizaci úsečky

Výsledkem této diplomové práce je applet, který demonstruje aplikaci Bresenhamova algorimu, popsaného v kapitole 4.2 výše. Jeho hlavním účelem je možnost otestovat si funkčnost algoritmu v praxi. Layout appletu je navržen tak, aby se studenti snadno zorientovali a seznámili s jeho ovládáním.

Pomocí dvou bodů, které umístíme kliknutím do plochy appletu, vytvoříme skutečnou úsečku, na základě které se podle zvoleného měřítka vykreslí takové pixely,



Obr. 6.1: Applet demonstrující Bresenhamův algoritmus.

abychom dostali co nejlepší výsledek rasterizace. Při práci s appletem můžeme měnit měřítko velikosti pixelů a detailněji tak prozkoumat princip Bresenhamova algoritmu. Vstupem pro algoritmus jsou dvě celočíselné souřadnice již zmíněných bodů, které definujeme kliknutím do vykreslovací plochy. To znamená, že následným pohybem již definovaných bodů v rámci jednoho stejného pixelu se vysvícené pixely nijak nezmění. V praxi je možné se setkat se situací, kdy budeme chtít zadat neceločíselné souřadnice, v takovém případě je potřeba Bresenhamův algoritmus upravit. V teoretické části je popsán způsob jak rasterizovat úsečku pro směrnici $0 < m < 1$, pro dalších 7 směrů je potřeba celý algoritmus zobecnit. Ukázkový applet již zobecněn je a je tedy možné úsečku vykreslit ve všech směrech.

6.3.1 Popis appletu pro rasterizaci úsečky

Applet je rozdělen na „kreslící plochu“ a ovládací panel ve spodu. Tlačítky „plus“ a „mínus“, případně posuvníkem, můžeme měnit měřítko velikosti pixelu, a to jak před zadaním koncových bodů, tak i po jejich zadání. Pokud jsme již zadali body a rozhodneme se měřítko změnit, automaticky se provede restart, a my tak můžeme kreslit znovu v novém měřítku. Zbývá tlačítko „reset“, které nám umožní zvolit nové dva koncové body pro vykreslení jiné úsečky. Nemusíme však body vždy znovu zadávat, můžeme editovat jejich polohu kliknutím na daný bod a tažením myši. Při puštění levého tlačítka myši bod umístíme. Rasterizace probíhá při každé změně bodů, a to v reálném čase.

Zaškrtávacím políčkem „zobrazit teoretickou čáru“ volíme zobrazení teoretické čáry, která je vykreslena červeně. Druhé políčko „zobrazit spojnici“ nám modře propojí body. Nejedná se však o teoretickou čáru, to by platilo jen v případě subpixelových souřadnic.

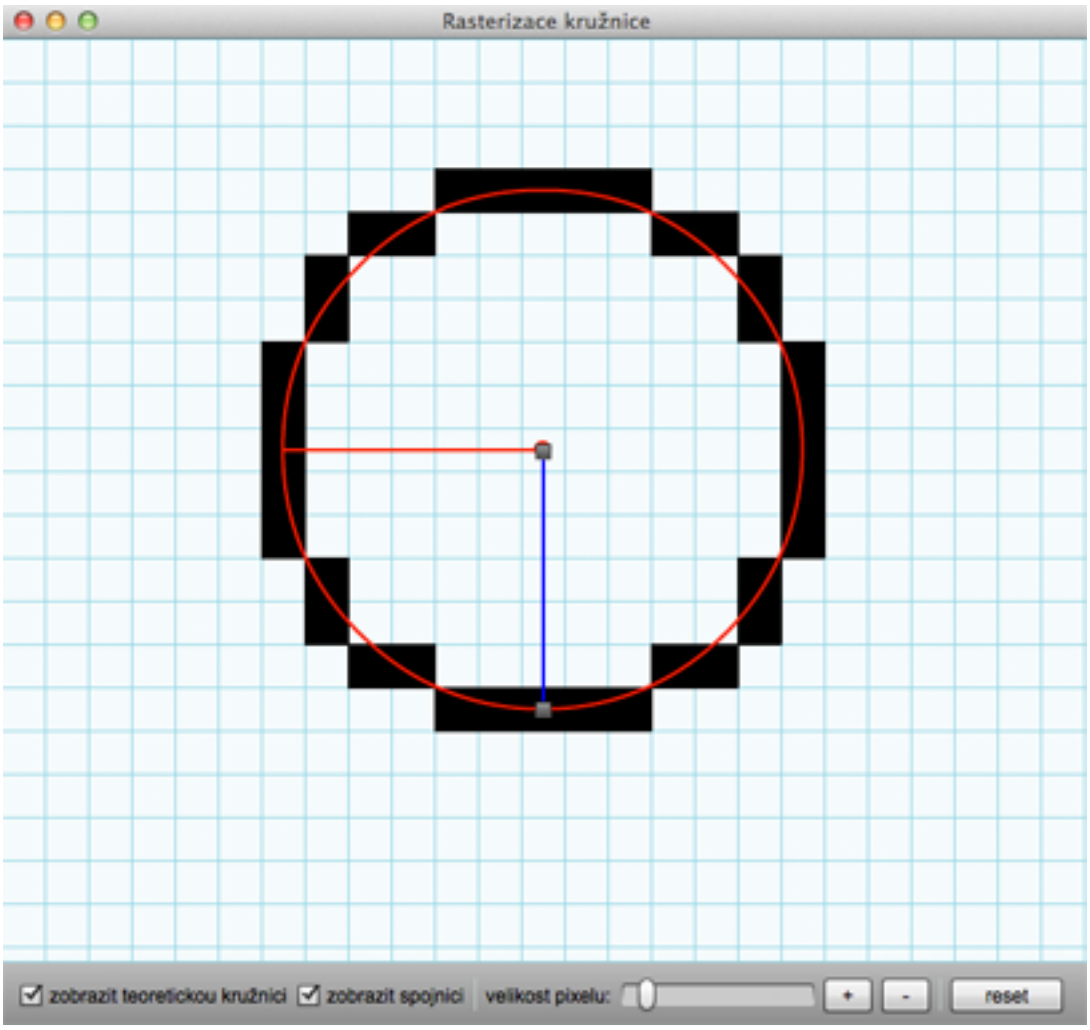
6.4 Applet pro rasterizaci kružnice

Dalším appletem je applet pro rasterizaci kružnice. Teorie k tomuto appletu je popsána v kapitole 4.4. Tento algoritmus, který vytvořil rovněž J. E. Bresenham, se nazývá Midpoint algoritmus. Pracuje na principu rozhodování, zda-li daný bod patří dovnitř či vně kružnice. Podle toho zvolíme jednu ze dvou možných souřadnic bodu, který leží vně kružnice. Protože kružnice je souměrná kolem počátečního bodu, nemusíme aplikovat algoritmus na všechny body, stačí jej aplikovat na jednu řídící osu, pro jeden směr a body transformovat do ostatních kvadrantů.

6.4.1 Popis appletu pro rasterizaci kružnice

Layout appletu je stejný jako u appletu pro rasterizaci úsečky. Ovládání je až na pár detailů také stejné. Jediný rozdíl je ve způsobu zadání kružnice. Postupujeme tak, že klikneme do pole, kde chceme mít střed kružnice. Poté zvolíme druhý bod, rovněž kliknutím myši. Tím definujeme poloměr r kružnice. Body lze dodatečně posouvat a tedy měnit jak polohu středu, tak poloměr.

Zaškrtávacím políčkem „zobrazit teoretickou kružnici“ zobrazujeme či skrýváme teoretickou kružnici, pomocí které algoritmus rozhoduje, který bod ze dvou možných vybrat. Můžeme vidět, že se skutečně vykreslily pouze body, které patří středem bodu vně kružnice. Červená čára od středu je definovaný poloměr r, modrá čára, kterou můžeme skrýt či zobrazit pomocí políčka „zobrazit spojnici“, spojuje definované body.



Obr. 6.2: Applet demonstrující Midpoint algoritmus.

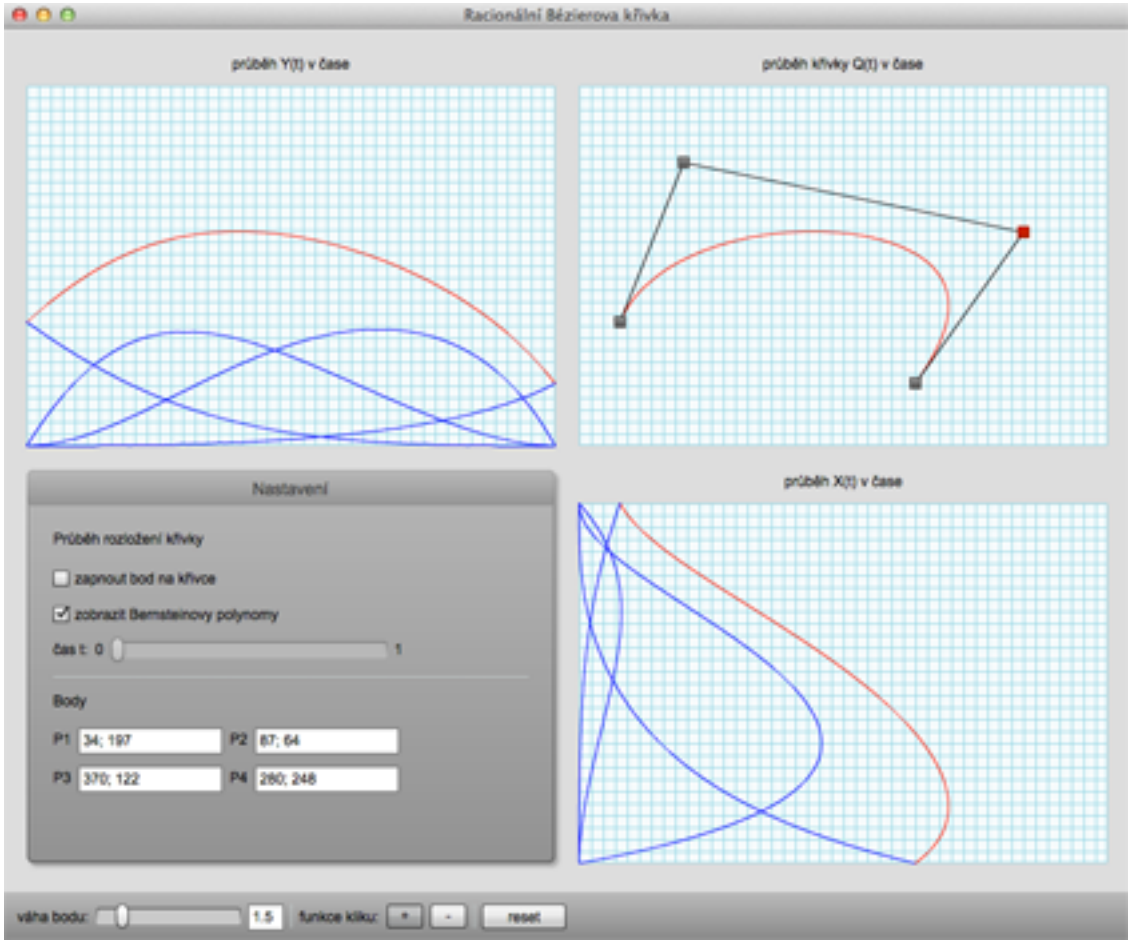
Měřítko velikosti pixelu můžeme měnit tlačítky „plus“ a „mínus“ , případně posuvníkem, a to jak před zadaním koncových bodů, tak i po jejich zadání. Tlačítkem „reset“ restartujeme aplikaci a můžeme opět začít kreslit.

6.5 Applet racionální Bézierova křivka

Applet umožňuje vykreslení racionálních Bézierových křivek třetího stupně, tedy pro 4 řídící body. Tyto křivky jsou popsány v kapitole 2.6. Jejich hlavní výhodou je možnost ovlivňování váhy (důležitosti) jednotlivých bodů, a tím dosahování vykreslení křivek, které by pomocí klasických Bézierových křivek nebyly dosažitelné, například kuželosečky. To je hlavním účelem tohoto appletu. Studenti si mohou vyzkoušet, jak změna váhy bodů ovlivňuje vykreslení a co se při změnách vah děje s

bázovými polynomy.

Vzorem pro tento applet je již vytvořený Java applet pro ukázku vykreslení Bézierových kubik. Applet má celkem tři grafy z nichž v jednom je vykreslena celá křivka $Q(t)$ a v dalších dvou je tato křivka rozdělena na průběhy $X(t)$ a $Y(t)$, ze kterých se $Q(t)$ skládá. Máme možnost vidět racionální Bersteinovy polynomy, jejichž součtem x-ových a y-ových souřadnic dostáváme křivku. Zapnutím bodu na křivce si můžeme snadno dokázat, že v bočních grafech se skutečně jedná o rozložení křivky.



Obr. 6.3: Applet pro racionální Bézierovu křivku.

6.5.1 Popis appletu pro racionální Bézierovu křivku

Applet má opět vykreslovací plochu, i když nyní je menší a umístěná v pravém hornám rohu pod nápisem „průběh křivky $Q(t)$ v čase“. Pokud chceme zvolit řídící body pro racionální Bézierovu kubiku, provedeme to kliknutím do vykreslovací plochy. Pro kubiku potřebujeme čtyři body, tedy klikneme čtyřikrát. Poté se nám

vykreslí zadaná křivka a ihned můžeme vidět její rozklad na již zmíněné části. Ovládací zaškrtávací tlačítka nám umožní zapínat a vypínat polynomy a bod na křivce. Nastavováním posuvníku „čas t“ určujeme pozici zobrazeného bodu na křivce od začátku do konce křivky, tedy pro čas $t \in \langle 0, 1 \rangle$. Pro tuto operaci však musíme mít zaškrtlé políčko „zapnout bod na křivce“.

Váhu bodů změníme tak, že klikneme na jeden z nich levým tlačítkem myši, bod se aktivuje a vybarví červeně. Posuvník vlevo dole se nám aktualizuje na váhu daného bodu a my ji můžeme měnit. Okamžitě tak vidíme, jak se celá křivka mění a s ní i samozřejmě racionální Bersteinovy polynomy.

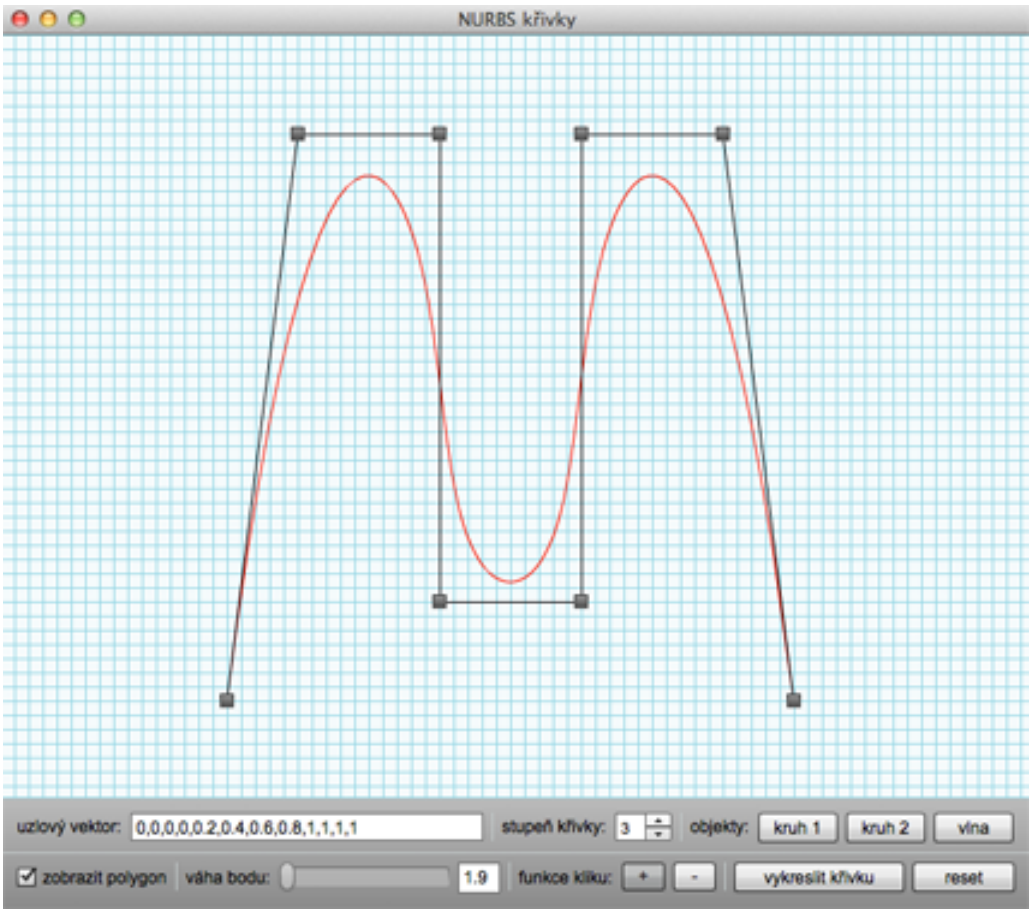
Pokud bychom chtěli některý bod odstranit, applet má dva módy práce s body. Jeden je vkládání a editace (tlačítko „plus“) a druhý odebírání (tlačítko „mínus“). Pro zmíněnou operaci tedy přepneme na mód odebrání a bod odebereme. Nová křivka se nám opět zobrazí až po vložení nového bodu. Pro odstranění všech bodů je zde tlačítko „reset“.

6.6 Applet NURBS křivky

Applet pro modelování NURBS křivek je nejsložitějším, ale také nejvariabilnějším, appletem z celé kolekce praktické části. Umožňuje studentovi vykreslit libovolnou NURBS křivku, libovolného stupně s libovolným počtem bodů a různými hodnotami jejich vah. Celý applet je napojen na třídu Nurbs.as, která implementuje rovnice z teoretické části. NURBS křivka v appletu je vykreslena pomocí B-spline polynomů. Jako daň za variabilnost appletu je třeba znát základní teorii NURBS křivek, která je popsána v příslušné části této diplomové práce. Abychom ovšem pozorovatele neunudili matematickými aparáty, applet je uzpůsoben tak, aby si i neznalý uživatel mohl vyzkoušet, jak NURBS křivky fungují. To je docíleno pomocí předpřipravených tvarů, kterými jsou „kružnice 1“, „kružnice 2“ a „vlna“. Všechny tyto tvary byly detailně rozebrány v kapitole 3.1.

6.6.1 Popis appletu pro NURBS křivku

Applety pro vykreslení křivek mají dva módy editace bodů. První je mód vkládání bodů a jejich editace (tlačítko „plus“) a druhý mód odebírání bodů (tlačítko „mínus“). Obě tlačítka jsou přepínací, takže může být nastaven buď jeden nebo druhý mód. Při aktivním módu vkládání klikem do vykreslovací plochy vložíme bod. Rovněž můžeme kliknout na již vložené body a označit je tím k editaci např. vah. Pokud je zapnut mód odebírání bodů, klikem do vykreslovací plochy se nestane nic a klikem na bod se daný bod odstraní.



Obr. 6.4: Applet pro NURBS křivku.

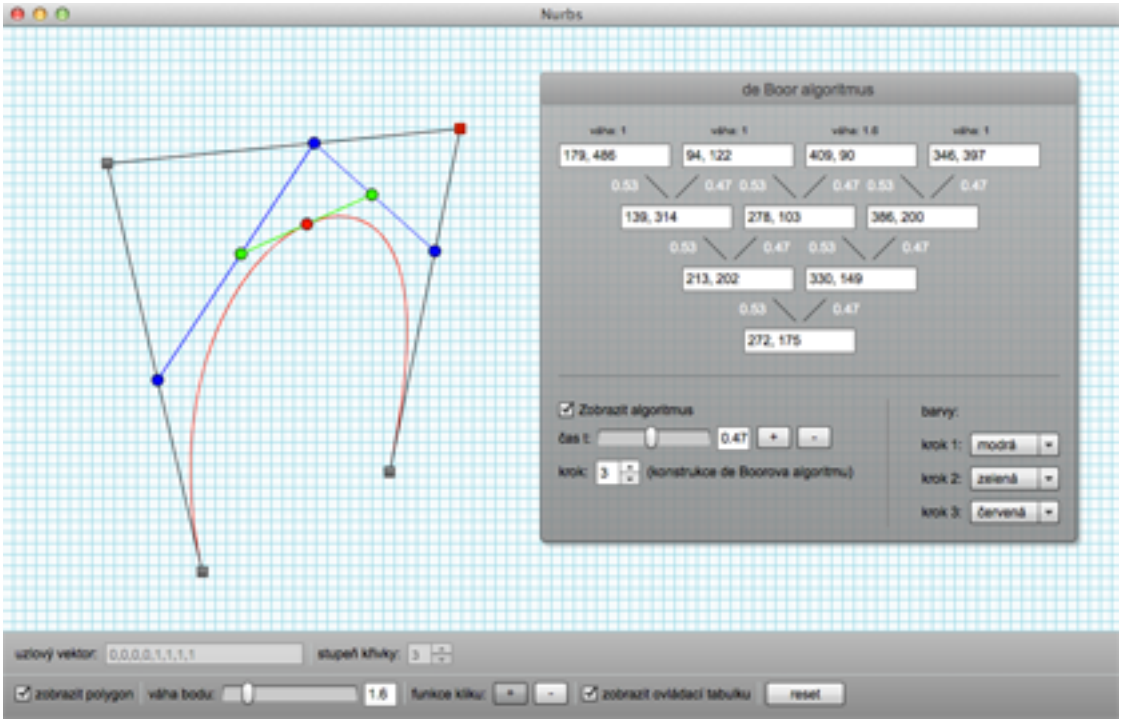
Kliknutím a tažením bodu v prvním režimu vkládání můžeme bod přemístit, křivka se přitom vykresluje v reálném čase se změnou bodu. Vyčistit vykreslovací plochu můžeme pomocí tlačítka reset, kterým vrátíme vše do počátečního stavu. U appletů, u kterých je možné kreslit libovolné množství bodů, např. tento applet, se křivka vykreslí buď kliknutím na tlačítko „vykreslit křivku“ nebo kliknutím na pravé tlačítko myši do vykreslovací plochy. Pravé tlačítko ovšem funguje jen u aplikací tedy nikoli na internetu v internetovém prohlížeči. Applet je udělaný tak, že se dá libovolně zvětšovat, takže při maximalizaci dosáhneme velké vykreslovací plochy. Checkbox „zobrazit polygon“ zobrazí nebo skryje konvexní obálku křivky.

Tento applet disponuje několika rozdíly oproti ostatním. Pomocí políčka „uzlový vektor“ jsme schopni editovat chování křivky. Je však třeba znát princip funkce uzlového vektoru. V opačném případě dojde k vykreslení nesmyslné křivky či dokonce k vykreslení jediného bodu v levém horním rohu appletu. Po správném nastavení hodnot můžeme křivku vykreslit (aktualizovat) pravým kliknutím myši do pole appletu nebo kliknutím na tlačítko „vykreslit křivku“.

Applet má dva módy vykreslování, a to automatický a manuální. Pokud klikneme na jedno z tlačítek pro předdefinovaný tvar, applet se přepne do stavu automatického vykreslování. V takovém případě lze měnit pouze polohy bodů a jejich váhy, popřípadě body odebírat, nelze však další přidat. Při kliknutí do pole v módu „přidávání bodů“ (tlačítko plus) se automaticky přepneme do módu manuálního, kde můžete přidávat i ubírat body dle libosti. V manuálním módu při přidávání a ubírání bodů se automaticky generuje uzlový vektor tak, aby splnil vstupní požadavky, tedy požadavek na počet bodů a stupeň křivky n.

6.7 Applet de Boorova algoritmu

Podobně jako již vytvořený applet algoritmu de Casteljau i applet de Boorova algoritmu simuluje konstrukci pro zjištění bodu na křivce pomocí tohoto zobecněného algoritmu.



Obr. 6.5: Applet pro simulaci vykreslení NURBS křivky pomocí de Boorova algoritmu.

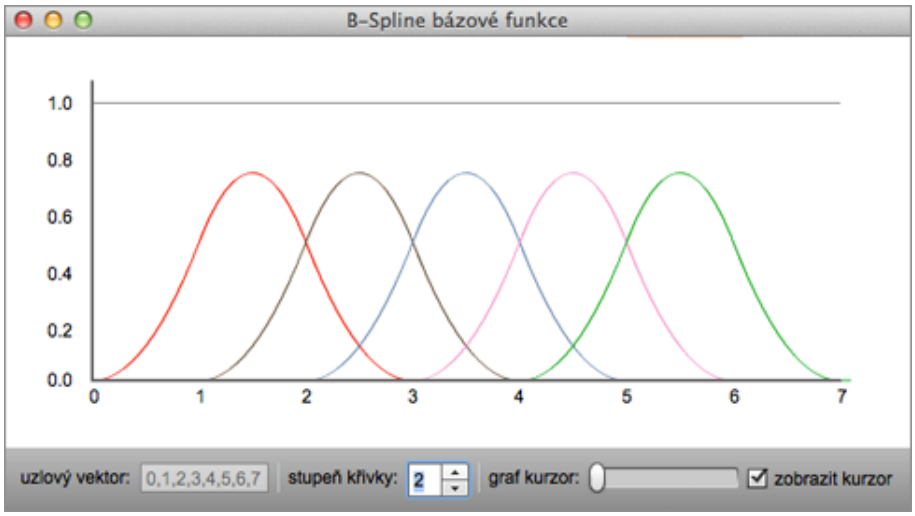
6.7.1 Popis appletu de Boorova algoritmu

Ovládání appletu je intuitivní. Krom již vysvětlených funkcí společných pro všechny applety má tento pár specifických funkcí, které jinde nenajdete. V tomto appletu

nelze ručně měnit uzlový vektor ani stupeň křivky. Pozornost nyní přesuňme k tabulce vpravo. Právě zde jsou zmíněné specifické funkce. Tabulku lze přemístit tažením myši, schovat či znovu otevřít kliknutím na checkbox „zobrazit ovládací tabulku“. V horní části tabulky je ukázán postup výpočtu výsledného bodu na křivce. V nejvyšší vrstvě jsou souřadnice řídících bodů. Nad každým políčkem řídícího bodu je zobrazena váha daného bodu. Tu lze upravovat tažením posuvníku vlevo dole na panelu. Každá další vrstva obsahuje o jedno políčko méně, vedle každé čáry je zobrazena aktuální hodnota proměnné α. V levé dolní části tabulky můžeme krokovat průběh konstrukce algoritmu, vypínat konstrukci či zapínat a měnit čas t, pro který chceme získat bod na křivce. To lze udělat tažením posuvníku nebo jemněji pomocí tlačítek plus a mínus. Barvy v pravo nám umožní změnit barvu jednotlivých kroků. Zaškrtávací políčko „zobrazit polygon“ zobrazí nebo skryje konvexní obálku křivky.

6.8 Applet B-spline bázové funkce

Tento applet vznikl jako doplněk k již vytvořeným NURBS appletům. Využívá výpočtu B-splinů třídy Nurbs.as a lehce předělané třídy Graph.as. Pomocí zvyšování stupně křivky (pomocí šipek) můžeme vidět jednotlivé B-spline bázové funkce.



Obr. 6.6: Applet zobrazující B-spline báze pro NURBS.

Jejich výpočet je dán uzlovým vektorem $U = \{0, 1, 2, 3, 4, 5, 6, 7\}$. Pro lepší prozkoumání je v appletu přidán do grafu vertikální kurzor, který nám pomůže určit amplitudu báze. Lze jím pohybovat pomocí posuvníku a zapínat zaškrtnutím checkboxu „zobrazit kurzor“.

7 ZÁVĚR

V teoretické části jsme probrali některá témata, která nám pomohou lépe chápat problematiku počítačové grafiky. Velkou část jsme věnovali různým druhům křivek a algoritmů pro jejich vykreslení. Výsledkem této diplomové práce je právě přehledně zpracovaná teoretická část a několik appletů. První applet simuluje problematiku rasterizace pomocí Bresenhamova algoritmu pro rasterizaci úsečky, druhý z této oblasti simuluje rasterizaci kružnice pomocí Midpoint algoritmu, třetí applet nás vtahuje do problematiky racionálních křivek, konkrétně racionálních Bézierových kubik. Pomocí čtvrtého appletu si můžeme vyzkoušet vykreslit NURBS křivku a pomocí editace váhových a uzlových vektorů pochopit, k čemu tyto vektory slouží a jak je vytvářet. Pátý de Boorův applet nám ukáže alternativu vykreslení NURBS pomocí numericky stabilního a velmi rychlého algoritmu. Poslední applet nám pro zadaný uzlový vektor zobrazí bázové funkce při zadání stupně křivky. Všechny Applety jsou navrženy tak, aby byly snadno pochopitelné a aby se snadno ovládaly. Pomocí těchto appletů si každý může prakticky vyzkoušet, jak vlastně různé algoritmy fungují, a lépe si tak představit mnohdy složitou problematiku počítačové grafiky.

V této práci jsem vytvořil všechny applety ze zadání kromě appletu pro adaptivní a neadaptivní rasterizaci. Nad rámec zadaných appletů jsem vytvořil applet pro zobrazení B-spline bázových funkcí a rovněž jsem naprogramoval i applet simulující Bresenhamův Midpoint algoritmus. Applet pro NURBS křivku je udělán velmi obecně a díky tomu v něm lze modelovat jakoukoli NURBS křivku s libovolným počtem řídících bodů, vah a volitelným uzlových vektorem. Všechny applety jsou nasazeny na příjemné grafické uživatelské rozhraní, které umožňuje editaci křivek a zobrazení výsledku v reálném čase. Jednotlivé applety jsou k dispozici jak na internetu přes webový prohlížeč, tak jako samostatné aplikace ke stažení - zamezí se tím potřeba použití internetového připojení.

Soubor těchto appletů je zkompletován pomocí přehledné webové prezentace s popisem základní teorie k danému tématu, podobně jak to bylo již realizováno v bakalářské práci. Webová prezentace je zpřístupněna přes internet pod hlavičkou VUT v Brně. Jednotlivé applety na sebe navazují a odkazují na sebe navzájem.

V příloze této práce nalezneme velmi podrobně rozepsaný kód základních a důležitých tříd. V rámci práce jsem naprogramoval velkou spoustu modifikací těchto základních tříd, neboť každý applet potřebuje jiné metody a proměnné. Nicméně všechny tyto třídy vycházejí ze stejného základu.

Všechny tyto applety svou funkčností slouží ke zlepšení úrovně výuky počítačové grafiky tak, že si student ověřuje probranou látku v praxi a platnost matematických rovnic a funkci algoritmů z oblasti počítačové grafiky.

LITERATURA

[1] Jiří Žára, Bedřich Beneš, Jiří Sochor, Petr Felkel *Moderní počítačová grafika* 2. přepracované a rozšířené vydání Computer Press, a.s., 2010 Brno. ISBN: 80-251-0454-0

[2] Chafic Kazoun a Joey Lott *Programming Flex 3* Copyright © 2008 O'Reilly Media O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472 ISBN: 978-0-596-51621-5

[3] Gerald Farin *Curves and surfaces for GAGD* Copyright © 2002 Academic Press O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472 ISBN: 1-55860-737-4.

[4] Wikipedia *Polohový vektor* [online]. poslední aktualizace 22. 6. 2011 [cit. 5. 12. 2011]. Dostupné z URL: <`http://cs.wikipedia.org/wiki/Polohový_vektor`>.

[5] Wikipedia *Adobe AIR* [online]. poslední aktualizace 8. 12. 2011 [cit. 11. 12. 2011]. Dostupné z URL: <`http://en.wikipedia.org/wiki/Adobe_AIR`>.

[6] Wikipedia *Rich Internet application* [online]. poslední aktualizace 3. 10. 2011 [cit. 11. 12. 2011]. Dostupné z URL: <`http://en.wikipedia.org/wiki/Rich_Internet_application`>.

[7] Wikipedia *AJAX* [online]. poslední aktualizace 24. 10. 2011 [cit. 11. 12. 2011]. Dostupné z URL: <`http://cs.wikipedia.org/wiki/AJAX`>.

[8] Wikipedia *HTML5* [online]. poslední aktualizace 7. 12. 2011 [cit. 11. 12. 2011]. Dostupné z URL: <`http://cs.wikipedia.org/wiki/HTML5`>.

[9] Adobe.com *Adobe AIR 3* [online]. poslední aktualizace 14. 7. 2009 [cit. 11. 12. 2011]. Dostupné z URL: <`http://www.adobe.com/cz/products/air.html`>.

[10] *Murphy's Modified Bresenham Line Drawing Algorithm* [online]. [cit. 14. 12. 2011]. Dostupné z URL: <`http://homepages.enterprise.net/murphy/thickline/index.html`>.

[11] ScottLogic *Flex, Silverlight or HTML5? Time to decide...* [pdf online]. [cit. 11. 12. 2011]. Dostupné z URL: <`http://www.scottlogic.co.uk/blog/colin/wp-content/uploads/2011/05/Flex-Silverlight-HTML5.pdf`>.

[12] Interval *Adobe Flex - co je a co není* [online]. poslední aktualizace 4. 6. 2008 [cit. 10. 12. 2011]. Dostupné z URL: <`http://interval.cz/clanky/adobe-flex-co-je-a-co-neni/`>.

[13] *Centre of Computer Graphics and Data Visualisation* poslední aktualizace 22. 11. 2011 [cit. 14. 12. 2011]. Dostupné z URL: <http://herakles.zcu.cz/education/ZPG/cviceni.php?no=11#part3/>.

SEZNAM SYMBOLŮ, VELIČIN A ZKRATEK

NURBS	Neuniformní racionální B-spline křivky
RIA	bohaté internetové aplikace – Rich Internet Application
MXML	značkovací jazyk založený na XML standartu
HTML	značkovací jazyk
DDA	Digital Differential Analyzer - rasterizační algoritmus
GUI	Grafické uživatelské rozhraní

SEZNAM PŘÍLOH

A OBSAH CD

Na přiloženém CD můžeme nalézt offline verzi webových stránek s Flash applety a stručnou teorií pro detailnější pochopení dané problematiky. Stránky se souborem appletů nalezneme ve složce „applety“. Spustit je můžeme pomocí souboru „index.html“ uvnitř složky. Stránky jsou rozčleněné do složek, kde každý applet je ve své složce, která nese jeho název. Uvnitř složky jsou soubory nutné pro funkci appletů a flash pluginu.

Zdrojové kódy všech tříd je možné najít pod odkazem „stáhnout“ v hlavním menu. Je možné si zde vybrat zdali stáhnout celý projekt pro program Flash Builder nebo jen textové soubory se zdrojovým kódem.

Pro spuštění appletů je třeba mít nainstalovaný Adobe Flash a pro standalone aplikace prostředí Adobe Air. Vše je zdarma ke stáhnutí na stránce Adobe.com.

Dále na přiloženém CD můžeme nalézt elektronickou verzi diplomové práce, která je shodná s vytištěnou a svázanou verzí. Najdeme ji v kořenovém adresáři ve formátu pdf nazvanou „diplomová práce“.

B VÝPIS KÓDŮ

B.1 Hlavnín aplikace

Tento kód patří hlavní aplikaci MXML, která je hlavní a rodičovskou třídou pro všechny ostatní třídy. Konkrétně tento kód patří appletu NURBS. Ostatní applety mají kód podobný, ne však stejný.

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication xmlns:fx="http://ns.adobe.com/mxml/2009"
                                        xmlns:s="library://ns.adobe.com/flex/
                                            spark"
                                        xmlns:mx="library://ns.adobe.com/flex/mx
                                            "
                                        xmlns:local="*"
                                        xmlns:core="core.*"
                                        creationComplete="init()"
                                        showStatusBar="false"
                                        minWidth="600" minHeight="620"
                                        width="730" height="620"
                                        title="NURBS křivky">

<fx:Metadata>
        [Event(name="pointStartDrag", type="events.BeginPointDrag")]
</fx:Metadata>

<fx:Script>
<![CDATA[
import algorithm.Nurbs;
import core.MyPoint;
import events.BeginPointDrag;
import mx.collections.ArrayList;
import org.osmf.events.TimeEvent;

private function init():void {
        _points = new Array();
        _weights = new Array();
        _knot = new Array();
        _nurbs = new Nurbs();
        _timer = new Timer(40);

        _degree = 2;
        _pointCounter = 0;
        _nurbs.step = 0.01;

        //event listeners
        addEventListener("PointDragEvent", pointEvent, true);
        myGraph.addEventListener(MouseEvent.MOUSE_DOWN, graphClickHandler, false);
        myGraph.addEventListener(MouseEvent.RIGHT_CLICK, graphRightClickHandler,
            false);
        _timer.addEventListener(TimerEvent.TIMER, timerHandler);

        //GUI
        myGraph.doubleClickEnabled = true;
        weightSlider.enabled = false;
```

```
        sdegree.value = _degree;
}

//--------------------------------------------------------------------------
//
//  Variables
//
//--------------------------------------------------------------------------

private var _points:Array;
private var _nurbs:Nurbs;
private var _pointDragging:Boolean = false;
private var _timer:Timer;
private var _pointCounter:int;
private var _pointToChangeWeight:int;
private var _weights:Array;
private var _knot:Array;
private var _degree:int;

//--------------------------------------------------------------------------
//
//  Methods
//
//--------------------------------------------------------------------------

/**
 * @private
 *
 * Event handler - leve kliknuti mysi
 */
private function graphClickHandler(e:Event):void {
if(e.target is MyPoint) {

        // projde vsechny body a najde
        // ten na ktery se kliklo a priradi mu
        // jmeno pro styl, ostatnim ji sebere
        for(var i:int = 0; i < _pointCounter; i++) {
                var p:MyPoint = myGraph.pointArrayList.getItemAt(i) as MyPoint;
                p.setStyle("styleName", "");
                if(p == e.target) {
                        _pointToChangeWeight = i;
                        p.setStyle("styleName", "selected");
                }
        }

        // nacte hodnoty vahy do posuvniku
        if(_pointToChangeWeight is Number) {
                weightSlider.value = _weights[_pointToChangeWeight];
                weightTI.text = _weights[_pointToChangeWeight].toString();
        }

        // mod odebirani bodu
        if(minus.selected) {
                _pointCounter--;

                // disabluje posubnik pro vahy
                if(weightSlider.enabled)
                        weightSlider.enabled = false;
```

```
                // vytvori novy vektor vah podle
                // aktualnich bodu
                var newWeights:Array = new Array();
                for(var w:int = 0; w < _weights.length; w++) {
                        if(w != _pointToChangeWeight)
                                newWeights.push(_weights[w]);
                }
                _weights = newWeights;

                // manualni mod kresleni
                // vytvori u vektor automaticky
                if(myGraph.drawMode == "manual") {
                var knots:Array = new Array();
                var k:int = _degree + 1;
                var s:int = (_pointCounter + k); //knot length
                var km:int = s - 2*k;
                var pk:int = (_pointCounter-1)/_degree;
                for(var j:int = 0; j < s; j++) {
                        if(j < k)
                                knots[j] = 0;
                        else if(j >= k && j <= (s-k-1)) {
                        for(var m:int = 1; m <= km; m++) {
                                if(j >= ((k + m*_degree) - _degree) && j < (k + m*
                                    _degree))
                                knots[j] = (m/pk);
                        }
                        }
                        else
                                knots[j] = 1;
                }
                tknot.text = knots.toString();
        }
        myGraph.removePoint(_pointToChangeWeight);
        }
        else {
                if(!weightSlider.enabled)
                        weightSlider.enabled = true;
        }

}
else {

        if(plus.selected) {
        // pokud je nastaven mod kresleni auto
        // resetni - aby tam nezustavaly
        // preddefinovane tvary
        if(myGraph.drawMode == "auto") {
                restart();
        }

        // nastavi manualni kresleni
        myGraph.drawMode = "manual";

        // citac bodu
        _pointCounter++;

        // vlozi bod do grafu
        myGraph.addPoint(mouseX, mouseY);
```

```
        // vytvori vahy
        _weights = [];
        for(var n:int = 0; n < _pointCounter; n++) {
                _weights[n] = 1.0;
        }

        //opet vytvori u vektor
        if(myGraph.drawMode == "manual") {

                knots = new Array();
                k = _degree + 1;

                s = (_pointCounter + k); //knot length
                km = s - 2*k;
                pk = (_pointCounter-1)/_degree;
                for(j = 0; j < s; j++) {
                        if(j < k)
                                knots[j] = 0;
                        else if(j >= k && j <= (s-k-1)) {
                        for(m = 1; m <= km; m++) {
                                if(j >= ((k + m*_degree) - _degree) && j < (k + m*
                                    _degree))
                                knots[j] = (m/pk);
                        }
                        }
                        else
                                knots[j] = 1;
                }

                tknot.text = knots.toString();
        }
}
}
}

/**
 * @private
 *
 * Event handler - prave tlacitko mysi
 */
private function graphRightClickHandler(e:MouseEvent):void {
        if(_pointCounter >= 2) {
                drawCurve();
        }
        else {
                restart();
        }
}

/**
 * @private
 *
 * Event handler - dragovani bodu
 */
private function pointEvent(e:BeginPointDrag):void {
        e.stopPropagation();
        if(e.action == "startDrag") {
                _timer.start();
                _pointDragging = true;
```

```
        }
        else if(e.action == "stopDrag") {
                _timer.stop();
                _pointDragging = false;
        }
}

/**
 * @private
 *
 * Event handler - jednou za 40 ms
 */
private function timerHandler(e:TimerEvent):void {
        if(_pointDragging)
                drawCurve();
}

/**
 * @private
 *
 * nastaveni vahoveho posuvniku
 */
private function weightSliderChangeHandler(e:Event):void {
        _weights[_pointToChangeWeight] = weightSlider.value;
        weightTI.text = weightSlider.value.toString();
        drawCurve();
}

/**
 * @private
 *
 * zobrazi polygon
 */
private function showVirtualLine():void {
        myGraph.showPolynom = showVirtualLineButton.selected;
        myGraph.polygon();
}

/**
 * @private
 *
 * vykresli krivku
 */
private function drawCurve():void {
        var al:ArrayList = myGraph.pointArrayList;
        var p:MyPoint;

        for(var i:int = 0; i < al.length; i++) {
                p = al.getItemAt(i) as MyPoint;
                _points[i] = new Point(p.x, p.y);
        }

        if(myGraph.drawMode == "manual") {
                _knot = [];
                var kn:Array = tknot.text.split(",");
                for(var k:int = 0; k < kn.length; k++) {
                        _knot[k] = parseFloat(kn[k]);
                }
        }
```

```
        // nastaveni pro algoritmus NURBS
        _nurbs.maxNumPoints = _pointCounter;
        _nurbs.points = _points;
        _nurbs.weights = _weights;
        _nurbs.knot = _knot;
        _nurbs.degree = _degree;

        // ziskani dat z algoritmu a vykresleni
        myGraph.curvePoints = _nurbs.algorithm();
        myGraph.drawCurve();
        myGraph.redrawPolynom();
}

/**
 * @private
 *
 * restart
 */
private function restart():void {
        _pointCounter = 0;
        _weights = [];
        myGraph.restart();
        weightSlider.value = 0;
        weightSlider.enabled = false;
}

/**
 * @private
 *
 * Event handler - plus click
 */
private function plusClickHandler(e:MouseEvent):void {
        if(plus.selected) {
                minus.selected = false;
        }
}

/**
 * @private
 *
 * Event handler - minus click
 */
private function minusClickHandler(e:MouseEvent):void {
        if(minus.selected) {
                plus.selected = false;
        }
}

/**
 * @private
 *
 * Event handler - stupen krivky (zmena)
 */
private function nsChangeHandler(e:Event):void {
        if(sdegree.value >= 2) {
                _degree = sdegree.value;
        }
}
```

```
/**
 * @private
 *
 * Tvar - kruh 1
 */
private function addCircle(e:MouseEvent):void {
        restart();

        // zadani bodu, u vektoru, w vektoru pro kruh1
        _pointCounter = 9;
        _knot = [0,0,0,1/4,1/4,2/4,2/4,3/4,3/4,1,1,1];
        _weights = [1.0,0.7071,1.0,0.7071,1.0,0.7071,1.0,0.7071,1.0];
        _degree = 2;

        sdegree.value = 2;
        tknot.text = _knot.toString();

        myGraph.addPoint(160, 270);
        myGraph.addPoint(160, 470);
        myGraph.addPoint(360, 470);
        myGraph.addPoint(560, 470);
        myGraph.addPoint(560, 270);
        myGraph.addPoint(560, 70);
        myGraph.addPoint(360, 70);
        myGraph.addPoint(160, 70);
        myGraph.addPoint(160, 270);

        myGraph.drawMode = "auto";
        drawCurve();
}

/**
 * @private
 *
 * Tvar - kruh 2
 */
private function addCircle2(e:MouseEvent):void {
        restart();

        // zadani bodu, u vektoru, w vektoru pro kruh2
        _pointCounter = 7;
        _knot = [0,0,0,1/4,1/2,1/2,3/4,1,1,1];
        _weights = [1.0,0.5,0.5,1.0,0.5,0.5,1.0];
        _degree = 2;

        sdegree.value = 2;
        tknot.text = _knot.toString();

        myGraph.addPoint(160, 270);
        myGraph.addPoint(160, 470);
        myGraph.addPoint(560, 470);
        myGraph.addPoint(560, 270);
        myGraph.addPoint(560, 70);
        myGraph.addPoint(160, 70);
        myGraph.addPoint(160, 270);

        myGraph.drawMode = "auto";
        drawCurve();
```

```
}

/**
 * @private
 *
 * Tvar - vlna
 */
private function addWave(e:MouseEvent):void {
        restart();

        // zadani bodu, u vektoru, w vektoru pro vlnu
        _pointCounter = 8;
        _knot = [0,0,0,0,1/5,2/5,3/5,4/5,1,1,1,1];
        _weights = [1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0];
        _degree = 3;

        sdegree.value = 3;
        tknot.text = _knot.toString();

        myGraph.addPoint(160, 470);
        myGraph.addPoint(210, 70);
        myGraph.addPoint(310, 70);
        myGraph.addPoint(310, 400);
        myGraph.addPoint(410, 400);
        myGraph.addPoint(410, 70);
        myGraph.addPoint(510, 70);
        myGraph.addPoint(560, 470);

        myGraph.drawMode = "auto";
        drawCurve();
}

]]>
</fx:Script>


<s:VGroup gap="0" width="100%" height="100%">
<core:Graph id="myGraph" width="100%" height="87%"/>
<s:SkinnableContainer id="bar" width="100%" height="13%">
        <s:VGroup width="100%">
        <s:HGroup paddingLeft="12">
        <s:Label text="uzlový vektor:" verticalAlign="middle" height="22"/>
        <s:TextInput id="tknot" text="" width="249" height="20" />
        <mx:VRule height="20" width="1"/>
        <s:Label text="stupeň křivky:" verticalAlign="middle" height="22"/>
        <s:NumericStepper id="sdegree" height="20" width="40" fontSize="11" minimum
            ="2" change="nsChangeHandler(event)"/>
        <mx:VRule height="20" width="1"/>
        <s:Label text="objekty:" verticalAlign="middle" height="22"/>
        <s:Button label="kruh 1" width="56" click="addCircle(event)"/>
        <s:Button label="kruh 2" width="56" click="addCircle2(event)"/>
        <s:Button label="vlna" width="56" click="addWave(event)"/>
        </s:HGroup>
        <mx:HRule x="0" y="575" width="100%" height="1"/>
        <s:HGroup paddingLeft="12">
        <s:CheckBox id="showVirtualLineButton" height="20" label="zobrazit polygon"
             change="showVirtualLine()" selected="true"/>
        <mx:VRule height="20" width="1"/>
        <s:Label text="váha bodu:" verticalAlign="middle" height="24"/>
```

```
        <s:HSlider id="weightSlider" width="120" height="20" minimum="0.0" maximum
            ="10.0" stepSize="0.1"
                        change="weightSliderChangeHandler(event)" verticalCenter
                            ="0"/>
        <s:TextInput id="weightTI" text="1.0" width="30"/>
        <mx:VRule height="20" width="1"/>
        <s:Label text="funkce kliku:" verticalAlign="middle" height="24"/>
        <s:ToggleButton id="plus" width="30" height="20" label="+" click="
            plusClickHandler(event)" selected="true"/>
        <s:ToggleButton id="minus" width="30" height="20" label="-" click="
            minusClickHandler(event)"/>
        <mx:VRule height="20" width="1"/>
        <s:Button label="vykreslit křivku" width="120" click="
            graphRightClickHandler(event)"/>
        <s:Button label="reset" width="72" click="restart()"/>
        </s:HGroup>
        </s:VGroup>
</s:SkinnableContainer>
</s:VGroup>

</s:WindowedApplication>
```

B.2 Nurbs.as

Nurbs.as je ActionScriptovou třídou, která zajišťuje výpočet B-spline funkcí pro NURBS křivky. Nejprve musíme vytvořit instanci této třídy, poté ji nastavit vstupní data a v zápětí můžeme nechat spočítat body na křivce pomocí metody algorithm(), která nám vrátí pole bodů.

```
////////////////////////////////////////////////////////////////////////////////
//
//  VUT Brno Ústav telekomunikací
//  Copyright © 2012
//  Všechna práva vyhrazena.
//
////////////////////////////////////////////////////////////////////////////////

package algorithm
{
import flash.geom.Point;
import flash.utils.ByteArray;

public class Nurbs
{

//--------------------------------------------------------------------------
//
//  Static
//
//--------------------------------------------------------------------------

//--------------------------------------------------------------------------
//
```

```
//  Constructor
//
//--------------------------------------------------------------------------

public function Nurbs() {
        init();
}

private function init():void {
        _points = new Array();
}

//--------------------------------------------------------------------------
//
//  Variables
//
//--------------------------------------------------------------------------

//--------------------------------------------------------------------------
//
//  Properties
//
//--------------------------------------------------------------------------

//----------------------------------
//  points
//----------------------------------

/**
 * @private
 */
private var _points:Array;

/**
 *
 */
public function get points():Array
{
        return _points;
}

/**
 * @private
 */
public function set points(value:Array):void
{
        _points = new Array();
        for each(var p:Point in value) {
                _points.push(p);
        }
}

//----------------------------------
//  weights
//----------------------------------

/**
 * @private
 */
```

```
private var _weights:Array;

/**
 *
 */
public function get weights():Array
{
        return _weights;
}

/**
 * @private
 */
public function set weights(value:Array):void
{
        _weights = new Array();
        for each(var n:Number in value) {
                _weights.push(n);
        }
}

//---------------------------------
//  curve points
//---------------------------------

/**
 * @private
 */
private var _curvePoints:Array;

/**
 *
 */
public function get curvePoints():Array
{
        return _curvePoints;
}

/**
 * @private
 */
public function set curvePoints(value:Array):void
{
        for each(var p:Point in value) {
                _curvePoints.push(p);
        }
}

//---------------------------------
//  maxNumPoints
//---------------------------------

/**
 * @private
 */
private var _maxNumPoints:int;

/**
 *
```

```
 */
public function get maxNumPoints():int
{
        return _maxNumPoints;
}

/**
 * @private
 */
public function set maxNumPoints(value:int):void
{
        _maxNumPoints = value;
}

//----------------------------------
//  step
//----------------------------------

/**
 * @private
 */
private var _step:Number;

/**
 *
 */
public function get step():Number
{
        return _step;
}

/**
 * @private
 */
public function set step(value:Number):void
{
        _step = value;
}

//----------------------------------
//  knot - u vector
//----------------------------------

/**
 * @private
 */
private var _knot:Array;

/**
 *
 */
public function get knot():Array
{
        return _knot;
}

/**
 * @private
 */
```

```
public function set knot(value:Array):void
{
        _knot = value;
}

//----------------------------------
//  degree - stupen krivky
//----------------------------------

/**
 * @private
 */
private var _degree:int;

/**
 *
 */
public function get degree():int
{
        return _degree;
}

/**
 * @private
 */
public function set degree(value:int):void
{
        _degree = value;
}

//--------------------------------------------------------------------------
//
//  Methods
//
//--------------------------------------------------------------------------

public function algorithm():Array {

        var t:Number = 0.0; // time belongs <0,1>
        var k:int = _degree + 1;

        _curvePoints = new Array(); // vycisteni vrstvy bodu

        while(t <= 1.0) {
                _curvePoints.push(getPoint(t, k, _maxNumPoints-1, _weights, _points
                    , _knot)); //polynoms
                t += _step; //increment t by defined step
        }

        return _curvePoints;
}

/**
 * @private
 *
 * vrati bod na krivce v zadanem case
 */
private function getPoint(t:Number, n:int, m:int, w:Array, cpoints:Array, u:Array,
    wayToGetResult:String = "polynoms"):Point {
```

```
        if(t == 0) { //starts at first point of polygon
                return _points[0];
        }
        else if(t > 1-_step || t == 1.0) { //if it is last point, curve will end in
             last point of polygon
                return _points[_maxNumPoints-1];
        }
        else {

        if(wayToGetResult == "polynoms") {

                var point:Point = Q(t, n, m, u, w, cpoints);
                return point;

        }
        else if(wayToGetResult == "De Boor algorithm") {

                var pr:Point;
                var points:Array = new Array();
                for(var i:int = 0; i < cpoints.length; i++) {
                        points[i] = new Array();
                        for(var ii:int = 0; ii < cpoints.length; ii++) {
                                points[i][ii] = new Array();
                        }
                        pr = cpoints[i] as Point;
                        points[0][i][0] = pr.x * w[i];
                        points[0][i][1] = pr.y * w[i];
                        points[0][i][2] = w[i];
                }

                i = 0;

                while (t >= u[i]) i++;

                var a:Number;

                for(var j:int = 1; j <= i; j++) {
                        for(var k:int = (i-1-degree+j); k <= (i-1); k++) {
                        a = (t - u[k])/(u[k+degree+1-j] - u[k]);
                        for(var l:int = 0; l < 3; l++) {
                                points[j][k][l] = a * points[j-1][k][l] + (1-a) *
                                    points[j-1][k-1][l];
                        }
                        }
                }

                return new Point(points[degree][degree][0] / points[degree][degree
                    ][2], points[degree][degree][1] / points[degree][degree][2]);

        }

        }

        return null;

}

/**
```

```
 * @private
 *
 * vrati hodnotu bazove funkce
 * pro zadane parametry
 */
private function N(t:Number, k:int, i:int, u:Array):Number {

        var nni1:Number = 0.0;
        var nni2:Number = 0.0;
        var nn1:Number = 0.0;
        var nn2:Number = 0.0;

        var nni3:Number = 0.0;

        if(k == 1) {
                if(t >= u[i] && t < u[i+1])
                        return 1.0;
                else
                        return 0.0;
        }

        if ((u[i] < u[i+k+1]) || (i >= 0 && i <= _maxNumPoints-1)) {  // check for
            divide by zero
                nn1 = N(t, k-1, i, u);
                nn2 = N(t, k-1, i+1, u);
                if(nn1 == 0) nni1 = 0; else if((u[i+k-1] - u[i]) == 0) nni1 = 0;
                    else nni1 = ((t - u[i]) / (u[i+k-1] - u[i])) * nn1;
                if(nn2 == 0) nni2 = 0; else if((u[i+k] - u[i+1]) == 0) nni2 = 0;
                    else nni2 = ((u[i+k] - t) / (u[i+k] - u[i+1])) * nn2;
                nni3 = nni1 + nni2;
                return nni3;
        }
        else
                return 0.0;

}

/**
 * @private
 *
 * dilci vypocet N
 */
private function R(t:Number, k:int, i:int, m:int, u:Array, w:Array):Number {

        var d:Number = 0.0;
        var nni:Number = 0.0;
        var rni:Number = 0.0;

        for(var j:int = 0; j <= m; j++) {
                nni = N(t, k, j, u);
                d += w[j] * nni;
        }

        rni = (w[i] * N(t, k, i, u)) / d;
        return rni;

}

/**
```

```
 * @private
 *
 * vrati bod na krivce
 * pro zadane parametry
 */
private function Q(t:Number, k:int, m:int, u:Array, w:Array, p:Array):Point {

        // t = time
        // k = degree of curve plus 1
        // m = number of points - 1
        // u = knot vector
        // w = weight vector
        // p = point vector

        var rni:Number = 0.0;
        var px:Number = 0.0;
        var py:Number = 0.0;

        for(var i:int = 0; i <= m; i++) {

                rni = R(t, k, i, m, u, w);
                px += p[i].x * rni;
                py += p[i].y * rni;

        }

        return new Point(px, py);

}
}
}
```

B.3 Graph.as

Tato třída se stará o vykreslení bodů, křivek a konvexních obálek. Také obstarává kliknutí myši či pohyb řídícího bodu. Tvoří základní GUI rozhranní.

```
////////////////////////////////////////////////////////////////////////////////
//
//  VUT Brno Ústav telekomunikací
//  Copyright © 2012
//  Všechna práva vyhrazena.
//
////////////////////////////////////////////////////////////////////////////////

package core
{
import flash.events.MouseEvent;
import flash.geom.Point;
import flash.utils.Dictionary;

import mx.collections.ArrayList;
import mx.controls.Button;
import mx.core.UIComponent;
```

```
import mx.graphics.SolidColor;
import mx.graphics.SolidColorStroke;

import spark.components.Group;
import spark.components.SkinnableContainer;
import spark.primitives.Rect;


public class Graph extends SkinnableContainer {

//--------------------------------------------------------------------------
//
//  Constructor
//
//--------------------------------------------------------------------------

public function Graph()
{
        super();
        init();
}

private function init():void {
        pointArrayList = new ArrayList(); // list bodu
        _points = new Array();
        _curve = new UIComponent(); // vrstva pro krivku
        _grid = new UIComponent(); // vrstva pro mrizku
        _convexEnvelop = new UIComponent(); // vrstva pro polygon
        _pointLayer = new Group(); // vrstva pro body

        pointCounter = 0; // pocet bodu
        _drawMode = "manual" //mod vykresleni
        _showPolynom = true;

        //GUI
        var background:Rect = new Rect();
        background.left = 0;
        background.top = 0;
        background.right = 0;
        background.bottom = 0;
        background.fill = new SolidColor(0xf5fbfc);
        background.stroke = new SolidColorStroke(0x91d7e3);

        addElement(background);
        addElement(_grid);
        addElement(_curve);
        addElement(_convexEnvelop);
        addElement(_pointLayer);
}

//--------------------------------------------------------------------------
//
//  Variables
//
//--------------------------------------------------------------------------

private var _step:int = 10; // pocet pixelu mezi mrizkami
private var _curve:UIComponent; // vrstva pro krivku
private var _grid:UIComponent; // vrstva pro mrizku
```

```
private var _convexEnvelop:UIComponent; // vrstva pro obalku
private var _pointLayer:Group; // vrstva pro body

//--------------------------------------------------------------------------
//
//  Properties
//
//--------------------------------------------------------------------------

//----------------------------------
//  points
//----------------------------------

/**
 * @private
 */
private var _points:Array;

/**
 *
 */
public function get points():Array
{
        return _points;
}

/**
 * @private
 */
public function set points(value:Array):void
{
        for each(var p:Point in value) {
                _points.push(p);
        }
}

//----------------------------------
//  curvePoints
//----------------------------------

/**
 * @private
 */
private var _curvePoints:Array;

/**
 *
 */
public function get curvePoints():Array
{
        return _curvePoints;
}

/**
 * @private
 */
public function set curvePoints(value:Array):void
{
        _curvePoints = new Array();
```

```
        for each(var p:Point in value) {
                _curvePoints.push(p);
        }
}

//----------------------------------
//  drawMode
//----------------------------------

/**
 * @private
 */
private var _drawMode:String;

/**
 *
 */
public function get drawMode():String
{
        return _drawMode;
}

/**
 * @private
 */
public function set drawMode(value:String):void
{
        _drawMode = value;
}

//----------------------------------
//  showPolygon
//----------------------------------

/**
 * @private
 */
private var _showPolynom:Boolean;

/**
 *
 */
public function get showPolynom():Boolean
{
        return _showPolynom;
}

/**
 * @private
 */
public function set showPolynom(value:Boolean):void
{
        _showPolynom = value;
}

//----------------------------------
//  others
//----------------------------------
```

```
public var pointCounter:int;
public var pointArrayList:ArrayList;

//--------------------------------------------------------------------------
//
//  Methods
//
//--------------------------------------------------------------------------

/**
 * zobrazi ridici polygon
 */
public function polygon():void {
        if(_showPolynom && pointArrayList.length >= 2) {
                redrawPolynom();
        }
        else {
                _convexEnvelop.graphics.clear();
        }
}

/**
 * skryje ridici polygon
 */
public function hidePolynom():void {
        _showPolynom = false;
        _convexEnvelop.graphics.clear();
}

/**
 * prekresli ridici polygon
 */
public function redrawPolynom():void {
        if(_showPolynom && pointArrayList.length >= 2) {
                _showPolynom = true;
                _convexEnvelop.graphics.clear();
                var point1:MyPoint;
                var point2:MyPoint;
                for(var i:int = 0; i < pointArrayList.length - 1; i++) {
                        point1 = pointArrayList.getItemAt(i) as MyPoint;
                        point2 = pointArrayList.getItemAt(i+1) as MyPoint;
                        drawLine(_convexEnvelop, new Point(point1.x, point1.y), new
                             Point(point2.x, point2.y), 2);
                }
        }
        else {
                _convexEnvelop.graphics.clear();
        }
}

/**
 * vykresli pixel
 */
public function drawPixel(xi:Number, yi:Number):void {
        graphics.lineStyle(0);
        graphics.beginFill(0x000000);
        graphics.drawRect(xi, yi, 5, 5);
        graphics.endFill();
}
```

```
/**
 * vykresli linku mezi body
 */
public function drawLine(layer:UIComponent, p1:Point, p2:Point, color:uint = 0
    x000000, thickness:int = 1):void {
        layer.graphics.lineStyle(thickness,color);
        layer.graphics.moveTo(p1.x, p1.y);
        layer.graphics.lineTo(p2.x, p2.y);
}

/**
 * vlozi ridici bod do grafu
 */
public function addPoint(x:Number, y:Number):void {
        var point:MyPoint = new MyPoint(x,y);
        pointArrayList.addItem(point);
        _pointLayer.addElement(point);
        pointCounter++;
        polygon();
}

/**
 * odstrani ridici bod z grafu
 */
public function removePoint(p:int):void {
        pointArrayList.removeItemAt(p);
        _pointLayer.removeElementAt(p);
        pointCounter--;
        polygon();
}

/**
 * vykresli krivku pro spocitane body
 */
public function drawCurve():void {
        if(pointArrayList.length >= 2) {
                var mp:MyPoint = pointArrayList.getItemAt(0) as MyPoint;
                var previousPoint:Point = new Point(mp.x, mp.y);
                _curve.graphics.clear();
                for each(var p:Point in _curvePoints) {
                        drawLine(_curve, previousPoint, p, 0xFF0000);
                        previousPoint = p;
                }
                _curve.invalidateDisplayList();
        }
        else {
                _curve.graphics.clear();
        }
}

/**
 * restart vsech vrstev
 */
public function restart():void {
        pointArrayList.removeAll();
        pointCounter = 0;
        _points = [];
        _curvePoints = [];
```

```
        _curve.graphics.clear();
        _pointLayer.removeAllElements();
        _convexEnvelop.graphics.clear();
}

/**
 * @override
 * pretizeni pro vykresleni mrizky
 */
override protected function updateDisplayList(w:Number, h:Number):void {

        _grid.graphics.lineStyle(1, 0x91d7e3);
        var pos:int = 0;
        while((pos + _step) < width) {
                pos = pos + _step;
                _grid.graphics.moveTo(pos, 0);
                _grid.graphics.lineTo(pos, height);
        }
        pos = 0;
        while((pos + _step) < height) {
                pos = pos + _step;
                _grid.graphics.moveTo(0, pos);
                _grid.graphics.lineTo(width, pos);
        }

        super.updateDisplayList(w, h);
}
}
}
```

B.4 MidpointAlgorithm.as

Třída MidpointAlgorithm.as je podobná jako třída Nurbs.as. Líší se několika metodami pro výpočet dat. Zde se snažíme získat body, které se mají vysvítit při rasterizaci kružnice.

```
////////////////////////////////////////////////////////////////////////////////
//
//  VUT Brno Ústav telekomunikací
//  Copyright © 2012
//  Všechna práva vyhrazena.
//
////////////////////////////////////////////////////////////////////////////////

package algorithm
{
import flash.events.MouseEvent;
import flash.geom.Point;
import mx.collections.ArrayCollection;
import mx.collections.ArrayList;
import mx.core.UIComponent;

public class MidpointAlgorithm {
```

```
//--------------------------------------------------------------------------
//
//  Constructor
//
//--------------------------------------------------------------------------

public function MidpointAlgorithm()
{
        super();
        init();
}

private function init():void {
        _points = new ArrayList();
        _pixelsToDraw = new Array();
        _step = 1;
}

//--------------------------------------------------------------------------
//
//  Variables
//
//--------------------------------------------------------------------------

/**
 * @private
 *
 * pole souradnic pixelu, ktere se maji vykreslit
 */
private var _pixelsToDraw:Array;

//--------------------------------------------------------------------------
//
//  Properties
//
//--------------------------------------------------------------------------

//----------------------------------
//  points
//----------------------------------

/**
 * @private
 */
private var _points:ArrayList;

/**
 *
 */
public function get points():ArrayList
{
        return _points;
}

/**
 * @private
 */
public function set points(value:ArrayList):void
```

```
{
        _points = value;
}

//----------------------------------
//  step
//----------------------------------

/**
 * @private
 */
private var _step:int;

/**
 *
 */
public function get step():int
{
        return _step;
}

/**
 * @private
 */
public function set step(value:int):void
{
        _step = value;
}

//--------------------------------------------------------------------------
//
//  Methods
//
//--------------------------------------------------------------------------

/**
 * vypocita a vrati pole souradnic pixelu
 */
public function getCirclePoints():Array {
        _pixelsToDraw = [];
        var y1:Number = 0;
        var x1:Number = 0;
        var yi:Number = 0;
        var xi:Number = 0;
        var yt:Number = 0;
        var xt:Number = 0;
        var dvex:Number = 0;
        var dvey:Number = 0;
        var p1:Number = 0;
        var pi:Number = 0;
        var start:Point = _points.getItemAt(0) as Point;
        var end:Point = _points.getItemAt(1) as Point;
        var px:Number = (Math.floor(start.x / step) + 0.5) * step;
        var py:Number = (Math.floor(start.y / step) + 0.5) * step;
        var r:Number = Math.round((Math.sqrt(Math.pow((px - end.x),2) + Math.pow((
            py - end.y),2)) / step));

        x1 = 0;
        y1 = r;
```

```
        dvex = 3;
        dvey = 2*r - 2;
        p1 = 1 - r;
        pi = p1;
        xi = x1;
        yi = y1;

        // podminka pro jednu osminu kruhu
        while(xi <= yi) {

        // postupne promitani pixelu do dalsich kvadrantu
        for(var i:int = 0; i < 8; i++) {
                if(i == 0) { xt = xi; yt = yi; }
                if(i == 1) { xt = yi; yt = xi; }
                if(i == 2) { xt = -yi; yt = xi; }
                if(i == 3) { xt = -xi; yt = yi; }
                if(i == 4) { xt = -xi; yt = -yi; }
                if(i == 5) { xt = -yi; yt = -xi; }
                if(i == 6) { xt = yi; yt = -xi; }
                if(i == 7) { xt = xi; yt = -yi; }
                _pixelsToDraw.push(new Point(xt + Math.floor(px/step), yt + Math.
                    floor(py/step)));
        }
        if(pi > 0) {
                pi = pi - dvey;
                dvey = dvey - 2;
                yi = yi - 1;
        }
        pi = pi + dvex;
        dvex = dvex + 2;
        xi = xi + 1;
        }
        return _pixelsToDraw;
}
}
}
```

B.5 MyPoint.as

Důležitá třída MyPoint zajišťuje pohodlnost editace křivek i rasterizačních teoretických úseček či kružnic. Drží v sobě interně informace o své velikosti a stavu a komunikuje s třídou Graph.as.

```
////////////////////////////////////////////////////////////////////////////////
//
//  VUT Brno Ústav telekomunikací
//  Copyright © 2012
//  Všechna práva vyhrazena.
//
////////////////////////////////////////////////////////////////////////////////

package core
{
```

```
import events.BeginPointDrag;
import flash.events.MouseEvent;
import spark.components.Button;
import spark.components.supportClasses.Skin;
import spark.components.supportClasses.SkinnableComponent;
import spark.primitives.Rect;

public class MyPoint extends Button
{
//--------------------------------------------------------------------------
//
//  Constructor
//
//--------------------------------------------------------------------------

public function MyPoint(x:Number, y:Number, width:Number = 10, height:Number = 10)
{
        super();
        init(x, y, width, height);
}

private function init(x:Number, y:Number, width:Number, height:Number):void {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        dragging = false;
        addEventListener(MouseEvent.MOUSE_DOWN, pointMouseDownHandler, false);
}

//--------------------------------------------------------------------------
//
//  Variables
//
//--------------------------------------------------------------------------

/**
 * @private
 *
 * Dragging flag - drzi interne informaci, zdali je bod dragovan ci ne
 */
private var dragging:Boolean;

//--------------------------------------------------------------------------
//
//  Methods
//
//--------------------------------------------------------------------------

/**
 * @private
 *
 * Event handler - stisknuti leveho tlacitka mysi
 */
private function pointMouseDownHandler(e:MouseEvent):void {
        addEventListener(MouseEvent.MOUSE_MOVE, pointMouseMoveHandler);
        addEventListener(MouseEvent.MOUSE_UP, pointMouseUpHandler);
        removeEventListener(MouseEvent.MOUSE_DOWN, pointMouseDownHandler);
}
```

```
/**
 * @private
 *
 * Event handler - pohyb mysi
 */
private function pointMouseMoveHandler(e:MouseEvent):void {
        var event:BeginPointDrag = new BeginPointDrag("PointDragEvent", "startDrag"
            , this);
        if(!dragging) {
                startDrag();
                dragging = true;
                dispatchEvent(event);
        }
}

/**
 * @private
 *
 * Event handler - pusteni leveho tlacitka mysi
 */
private function pointMouseUpHandler(e:MouseEvent):void {
        var event:BeginPointDrag = new BeginPointDrag("PointDragEvent", "stopDrag",
             this);
        removeEventListener(MouseEvent.MOUSE_MOVE, pointMouseMoveHandler);
        removeEventListener(MouseEvent.MOUSE_UP, pointMouseUpHandler);
        addEventListener(MouseEvent.MOUSE_DOWN, pointMouseDownHandler, false);
        if(dragging) {
                stopDrag();
                dragging = false;
                dispatchEvent(event);
        }
}

}
}
```

B.6 BeginPointDrag.as

Jediná třída typu Event, která nám umožňuje získat instanci bodu, který nás zajímá, při kliku a pohybu bodu po vykreslovací ploše.

```
////////////////////////////////////////////////////////////////////////////////
//
//  VUT Brno Ústav telekomunikací
//  Copyright © 2012
//  Všechna práva vyhrazena.
//
////////////////////////////////////////////////////////////////////////////////

// Event trida - nese v sobe instanci bodu a nazev provadene akce
// na zaklade toho jsme schopni provadet akce jako napriklad dragovani apod.

package events
```

```
{

import flash.events.MouseEvent;
import core.MyPoint;

public class BeginPointDrag extends MouseEvent
{
//--------------------------------------------------------------------------
//
//  Constructor
//
//--------------------------------------------------------------------------

public function BeginPointDrag(type:String, action:String=null, point:MyPoint=null,
     bubbles:Boolean=false, cancelable:Boolean=false)
{
    super(type, bubbles, cancelable);
        _point = point;
        _action = action;
}

/**
 * @private
 */
private function init():void {

}

//--------------------------------------------------------------------------
//
//  Variables
//
//--------------------------------------------------------------------------

//----------------------------------
//  point - instance bodu
//----------------------------------

/**
 * @private
 */
private var _point:MyPoint;

/**
 *
 */
public function get point():MyPoint
{
    return _point;
}

/**
 * @private
 */
public function set point(value:MyPoint):void
{
        _point = value;
}
```

```
//----------------------------------
//  action - nazev provadene akce
//----------------------------------

/**
 * @private
 */
private var _action:String;

/**
 *
 */
public function get action():String
{
        return _action;
}

/**
 * @private
 */
public function set action(value:String):void
{
        _action = value;
}

}
}
```